Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Organización Industrial

Problemas de rutas de vehículos: modelos, aplicaciones logísticas y métodos de resolución

Autor:

Benito Quintanilla, Ana

Tutor:

Sáez Aguado, Jesús

Estadística e Investigación Operativa

Valladolid, junio 2015.

Resumen

El problema de rutas de vehículos es uno de los más comunes de optimización. Aunque existen muchas variantes, el presente documento se centra en el Problema de Rutas de Vehículos Capacitado, cuyo objetivo es encontrar la mejor ruta a seguir por una flota de vehículos que tienen que visitar unos clientes para llevar a cabo labores de recogida y/o distribución.

Para resolver este problema, se plantearán diferentes modelos exactos y aproximados, y se establecerán comparaciones entre todos ellos a través de la aplicación de los mismos a una batería de problemas.

Abstract

The Vehicle Routing Problem is one of the most common optimization problems. Although there are many variations, this document focuses on the Capacitated Vehicle Routing Problem, which objective is to find the best route for fleet vehicles that have to visit some clients to complete the collection and/or distribution tasks.

To solve this problem, different exact and approximate models will be proposed, and a comparison will established through their practical application to a list of problems.

Palabras clave

Rutas de vehículos, modelos de programación entera, heurísticas, Algoritmo de Clarke & Wright, GRASP.

Key words

Vehicle routing, integer programming models, heuristics, Clarke & Wright's Algorithm, GRASP.

Índice

CAPÍTULO 1.

Introducción

1.1. Justificación

En muchas organizaciones, la gestión de las actividades de recogida y/o distribución constituye un importante problema de toma de decisiones. Con el paso de los años, las empresas se han dado cuenta de la repercusión de dicha actividad en sus costes totales, especialmente con la subida del petróleo.

Todas las empresas se han preguntado alguna vez: ¿cuántos vehículos necesito para cubrir la demanda de mis clientes con el menor coste posible? o, ¿en qué orden debo visitar a éstos de forma que se minimicen mis costes?

La gestión de rutas de la mayoría de las empresas se determina basándose en la experiencia del encargado de rutas, quien de manera intuitiva suele decidir cómo llevar a cabo la recogida y/o distribución. La informatización, automatización y optimización de dicha tarea, implicaría sin duda, una mejora en cuanto a la propia determinación y gestión de las rutas, así como una ventaja competitiva por disminución de costes.

1.2. Objetivo general

Por tanto, el Objetivo General (OG) del presente Trabajo Fin de Grado consiste en proponer diferentes métodos informatizados para resolver los problemas de rutas.

1.3. Objetivos específicos

Para cumplir este objetivo general, se van a tratar de alcanzar los siguientes Objetivos Específicos (OE):

- OE 1.- Realizar una breve revisión del origen del Problema de Rutas de Vehículos.

- OE 2.- Conocer en qué consiste el Problema de Rutas de Vehículos Capacitado y qué variantes existen.
- OE 3.- Destacar la importancia de los Problemas de Rutas de Vehículos desde el punto de vista económico.
- OE 4.- Proporcionar una clasificación general de los métodos de resolución para el CVRP.
- OE 6.- Exponer algunos de los métodos exactos y aproximados que permiten resolver los Problemas de Rutas de Vehículos Capacitado.
- OE 7.- Programar los métodos exactos y aproximados anteriores usando el software Xpress- Mossel.
- OE 8.- Comparar los distintos métodos de resolución.
- OE 9.- Obtener conclusiones acerca de qué método da las mejores soluciones.

CAPÍTULO 2.

Antecedentes: El Problema del Viajero

2.1. Justificación

Como se verá más adelante, los problemas de rutas de vehículos son una generalización de problema del viajero, para el caso en el que la capacidad es importante debido al tamaño de los ítems, y no se puede por tanto llevar a cabo el trayecto con un solo vehículo.

2.2. Origen

Los orígenes de este problema desde el punto de vista matemático se remontan a 1930 en Viena y Harvard. En la Universidad de Harvard, Merril Flood se interesó por el TSP cuando empezó a trabajar en la búsqueda de una ruta óptima para un autobús escolar. Mientras tanto, en Vienna, Karl Menger enunció lo que entonces se denominaba "El problema del mensajero": buscar el camino más corto que uniera un conjunto de puntos cuyas distancias entre todos ellos eran conocidas.

En las décadas de los 50 y de los 60, el problema se hizo muy popular y comenzaron a estudiarse problemas para un mayor número de ciudades.

2.3. Definición

El problema del viajero (TSP) es uno de los más famosos y estudiados. En este problema, un viajante de comercio tiene que visitar un conjunto de clientes o ciudades, y regresar después al punto de partida. Antes de salir, deberá decidir en qué orden recorrer dichas ciudades para minimizar la distancia total recorrida. Señalar que tiene que pasar una sola vez por cada ciudad.

Formalmente, el problema del viajante de comercio se define sobre un grafo $G = (N, A)$, con costes asociados a los arcos (A) y donde los clientes a visitar representan los nodos (N). De esta forma, se trata de encontrar el circuito

Hamiltoniano de coste mínimo, entendiendo como circuito Hamiltoniano en G aquel que pasa exactamente una vez por cada nodo.

2.4. Tipos

Se distinguen tres tipos de problemas dependiendo de la naturaleza de la red:

- Symmetric Traveling Salesman Problem (STSP): si el grafo es no dirigido (Figura 2). Se entiende por grafo no dirigido cuando el flujo a través de todos los arcos se permite en ambas direcciones, lo que se conoce como arcos no dirigidos (Figura 1). Esto último es equivalente a decir que la matriz de costes es simétrica.

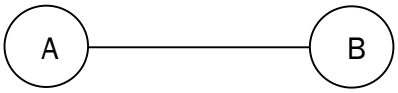

Figura 1. Arco no dirigido. (Elaboración propia).

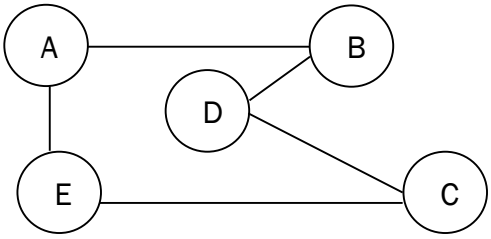

Figura 2. Grafo no dirigido. (Elaboración propia).

- Asymmetric Travelling Salesman Problem (ATSP): si el grafo es dirigido (Figura 4). Se entiende por grafo dirigido cuando el flujo a través de todos los arcos se permite sólo en una dirección, lo que se conoce como arcos dirigidos (Figura 3). Esto último es equivalente a decir que la matriz de costes no es simétrica.

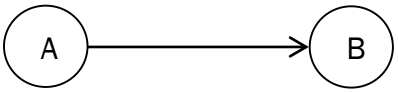

Figura 3. Arco dirigido. (Elaboración propia).

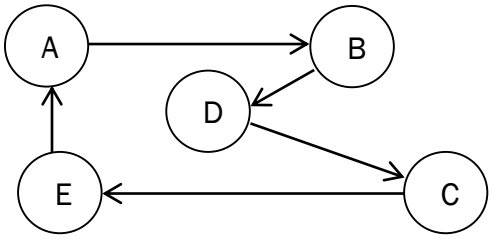

Figura 4. Grafo dirigido. (Elaboración propia).

- Mixtos: combinación de los dos casos anteriores.

Como todo grafo no dirigido puede transformarse en uno dirigido sustituyendo los arcos no dirigidos por un par de arcos dirigidos con direcciones opuestas (Figura 5), el STSP puede verse como un caso especial del ATSP.

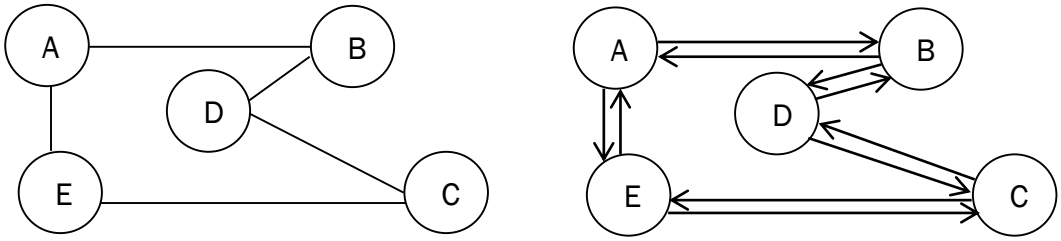

Figura 5. Transformación grafo no dirigido en grafo dirigido. (Elaboración propia).

2.5. Notación

- i hace referencia a cada nodo origen, donde $i = 1$ es el punto de partida.
- j hace referencia a cada nodo destino.
- n representa el número de nodos o clientes a visitar.
- x_{ij} indica si el viajante va o no de la ciudad i a la ciudad j.
- d_{ij} representa la distancia entre cada par de ciudades conectadas.
- y_{ij} representa el flujo de la ciudad i a la j.
- u_i representa el lugar que ocupa la ciudad i en la solución final.
- b_i representa la oferta (si es positiva) o la demanda (si es negativa) en cada nodo.

2.6. Formulación

2.6.1. Modelo inicial

$$Min \sum_{i=1}^{n} d_{ij} x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \dots, n \quad (1)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \dots, n \quad (2)$$

$$x_{ij} \in \{0, 1\} \quad i, j = 1, \dots, n$$

Donde:

(1) indica que de cada nodo sólo puede salir un arco.

(2) indica que a cada nodo sólo puede llegar un arco.

Formulado así, se trata de un problema de asignación, en el que la solución obtenida es una cota inferior de la distancia que el viajante va a recorrer. Además, dicha solución constará, generalmente, de varios subtours, dado que será más barato (Figura 6).

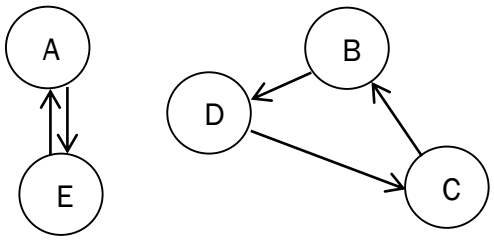

Figura 6. Formación de dos subtours. (Elaboración propia).

2.6.2. Restricciones de eliminación de subtours

Por tanto, para que el modelo nos proporcione una solución, hay que añadir restricciones para eliminar los subtours anteriormente mencionados.

Existen tres formas para ello:

a) Restricciones de eliminación de subtours

Se trata de incluir la condición de que haya al menos un arco que salga de cualquier subtour S:

$$\sum_{i \in S, j \in S} x_{ij} \geq 1, \ \ \forall S \subset N / S \neq N \quad (3)$$

El problema es que se generan demasiadas restricciones. Por ejemplo, si $|N| = n$, se necesitarían $(2^n - 2)$ restricciones del tipo anterior.

b) Restricciones de tipo Tucker-Miller-Zemlin.

Se trata de incluir la condición de que todo subtour debe contener al nodo inicial, lo que es equivalente a decir que sólo puede haber un subtour.

Para ello, se elige un nodo inicial y se añaden $(n-1)$ variables auxiliares $(u_i, i = 2, \dots, n)$ y las restricciones:

$$u_i - u_j + nx_{ij} \leq n - 1, \forall\, ij \,/\, 2 \leq i \neq j \leq n \quad (4)$$

Mediante reducción al absurdo se puede demostrar que añadiendo estas restricciones es imposible que se forme ningún subtour.

c) Problema auxiliar de redes

Consiste en asegurarnos que todos los nodos están conectados entre sí. Para ello, se añaden unas variables de flujo $y_{ij} \geq 0$ y se plantea un problema de flujo en redes con la siguiente formulación:

$$Min \sum_{i=1}^{n} d_{ij}x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \dots, n \quad (1)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \dots, n \quad (2)$$

$$\sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} y_{ji} = b_i \quad i = 1, \dots, n \quad (5)$$

$$y_{ij} \leq (n-1)x_{ij} \quad i, j = 1, \dots, n \quad (6)$$

$$x_{ij} \in \{0, 1\} \quad i, j = 1, \dots, n$$

$$y_{ij} \geq 0 \quad i, j = 1, \dots, n$$

Donde:

(5) se conoce como ecuación de conservación del flujo, ecuación de balance o ecuación de Kirchoff, e indica que el flujo de salida menos el flujo de entrada debe ser igual a la oferta o demanda en el nodo.

(6) relaciona la variable binaria y la variable de flujo, donde $(n-1)$ representa una cota superior.

Señalar que, en el nodo inicial se asigna oferta n, es decir, $b_1 = n$, mientras que para el resto de nodos se toma demanda 1, es decir, $b_i = -1$, $i = 2, \dots, n$.

2.7. Conexión con el Problema de Rutas de Vehículos

Tal y como se anticipó en el Apartado 2.1., el TSP ya es un primer caso de un Problema de Rutas cuando los objetos a transportar son pequeños, de forma que la cuestión de la capacidad no es un tema importante y por tanto, se puede llevar a cabo el transporte de la mercancía con un solo vehículo.

De hecho, existen heurísticas para resolver los Problemas de Rutas de Vehículos como "First Cluster then Routing", que lo que hacen es resolver primero un TSP, o gran tour, y luego lo dividen en varias rutas.

Además, los modelos exactos que se han visto para resolver el TSP (redes o mediante restricciones de tipo Tucker- Miller- Zemlin), se usan en los Problemas de Rutas de Vehículos Capacitado para mejorar las rutas obtenidas través de heurísticas. Esto, se verá en el Apartado 6.3.1.

CAPÍTULO 3.
Introducción al Problema de Rutas de Vehículos

3.1. Justificación

La trascendencia económica del sector del transporte genera costos de gran envergadura. La distribución física representa para las empresas entre la sexta y la cuarta parte de las ventas y entre uno y dos tercios del total de los costos logísticos (Ballou, 1991).

Además, una adecuada gestión de los problemas de distribución afecta directamente a la competitividad de las empresas. Así, el establecimiento de rutas y horarios para vehículos constituye un conjunto de problemas habituales que si no se resuelven de manera óptima, acarrean una merma significativa en la cuenta de resultados de las empresas. Autores como Kotler (1991) afirman que pueden conseguirse ahorros sustanciales en el área de la distribución física, y la describen como "la última frontera para obtener economías en los costos" y "el continente oscuro de la economía". Drucker (1962) describió las actividades logísticas que se llevaban a cabo tras la fabricación como las "áreas peor realizadas y a la vez más prometedoras dentro del mundo industrial". Incluso el recorte de una pequeña fracción de los costos de distribución puede aflorar enormes ahorros económicos, además de incrementar significativamente la satisfacción de los clientes.

Así, el Problema de Rutas de Vehículos resulta extremadamente útil en una gran variedad de casos reales. A continuación, se comentan brevemente algunas de sus aplicaciones más importantes:

- Logística.- Las aplicaciones más directas y más abundantes del VRP se centran en el campo de la logística. Entre ellas, se encuentran: rutas de vendedores, rutas escolares, reparto de mercancías y correo o sistema de recogidas de basura.
- Industria.- Las aplicaciones en industria no son tantas como en logística. Un ejemplo de dicha aplicación es la secuenciación de tareas.

3.2. Origen

Como ya se introdujo en el Apartado 2.8, el Problema de Rutas de Vehículos surgió como una extensión del TSP para el caso en el que es necesario realizar varias rutas.

En 1959, Dantzig y Ramser fueron los primeros autores que trataron este tema cuando estudiaron la distribución de gasolina para estaciones de carburante. Propusieron una formulación matemática para encontrar una forma de asignar los camiones a las estaciones de servicio de manera que se satisficieran las demandas de éstas y la distancia recorrida por la flota de camiones fuese la mínima.

Cinco años después, en 1964, Clarke y Wright propusieron el primer algoritmo que resultó efectivo para resolverlo y que se verá en el Apartado 6.1.

Desde entonces, se han planteado numerosos métodos de resolución, tanto exactos como aproximados, y se han desarrollado gran cantidad de páginas web, congresos, libros y aplicaciones comerciales dedicadas exclusivamente a este tema.

3.3. Definición

Los Problemas de Rutas de Vehículos (PRV, o VRP del inglés Vehicle Routing Problem) son una serie de problemas que hacen referencia al transporte de mercancías a ciertos puntos con una flota de vehículos. Formalmente, se definen sobre un grafo completo $G = (V, A)$, con costes asociados a los arcos (A) y donde los clientes representan los vértices (V), siendo el vértice 1 el depósito o almacén donde se encuentran los vehículos de transporte y la mercancía.

3.4. El problema básico y sus variantes

Aunque en este trabajo nos centraremos en el problema básico, llamado Problema de Rutas de Vehículos Capacitado (PRVC, o CVRP del inglés Capacitated Vehicle Routing Problem), cuya finalidad es determinar las rutas que una flota de vehículos, con capacidad dada y que salen de un punto común o depósito, debe seguir para cubrir la demanda de una serie de clientes, existen muchas extensiones de éste (se muestran las más usuales en aplicaciones reales. Figura 7):

- **VRP con ventanas de tiempo** (*time windows*) (VRPTW): el servicio a cada cliente tiene que realizarse en un intervalo horario determinado, teniendo que permanecer el vehículo en el lugar del cliente durante dicho servicio.

- **VRP con retornos** (*backhauls*) (VRPWB): también conocido como Linehaul-Backhaul Problem, en este tipo de problemas, los clientes se dividen en dos grupos. El primero de ellos, denominado *"linehaul"*, engloba aquellos clientes a los que hay que entregar una cantidad determinada de producto. El segundo, "*backhaul",* contiene a aquellos a los que hay que recoger una cantidad determinada de producto. La principal restricción es que para cada ruta, todas las entregas deben hacerse antes que las recogidas.
- **VRP multi- depósito**: los vehículos no son diáfanos, sino que tienen divisiones interiores. La mayoría de las veces, la dificultad de este problema reside en que cada producto tiene que ir en una de esas dependencias, no pudiéndose mezclar entre ellos.
- **VRP periódico** (VRPP): el servicio a los clientes no se realiza todos los días sino de forma periódica (cada $\boldsymbol{M}$ días).
- **VRP con inventarios** (VRPI): la cantidad demandada por cada cliente no es fija, sino que dependerá del nivel mínimo de inventario que éste quiera mantener.

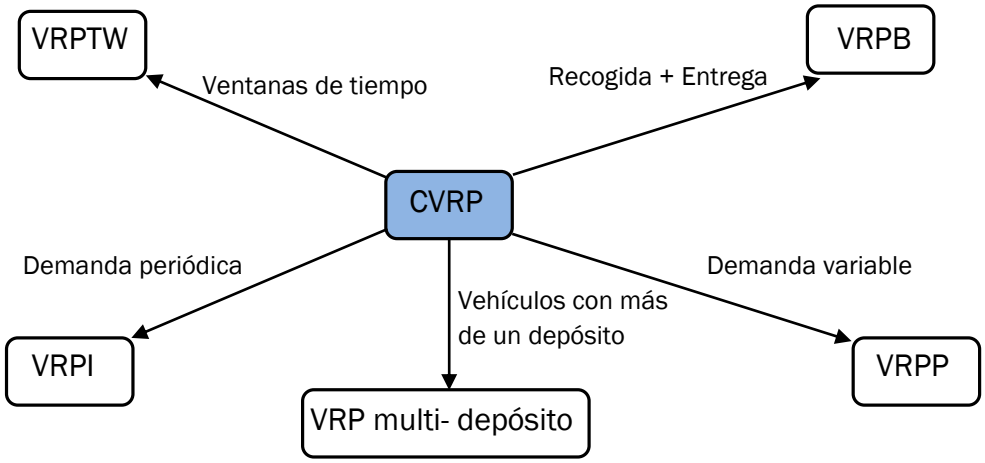

Figura 7. El CVRP y sus variantes. (Elaboración propia).

3.5. Complejidad

A pesar de la aparente sencillez de su planteamiento, es muy complejo de resolver: siendo n el número de clientes a visitar, el número total de posibles rutas es $(n-1)!/2$.

3.6. Tipos de Problemas de Rutas de Vehículos Capacitado

Se distinguen dos tipos de CVRP, en función de la naturaleza de la red:

- Problema de Rutas de Vehículos Capacitado Simétrico (SCVRP): si el grafo es no dirigido, es decir, si la matriz de costes es simétrica $(c_{ij} = c_{ji})$.
- Problema de Rutas de Vehículos Capacitado Asimétrico (ACVRP): si el grafo es dirigido, es decir, si la matriz de costes es asimétrica.

Al igual que se expuso en el Problema del Viajero, dado que todo grafo no dirigido puede transformarse en uno dirigido, el SCVRP puede verse como un caso particular del ACVRP.

3.7. Notación

- i hace referencia al nodo origen, donde $i = 1$ es el depósito.
- j hace referencia al nodo destino.
- n representa el número de nodos o clientes a visitar.
- h hace referencia al vehículo.
- K representa el número de vehículos.
- x_{ij}indica si el arco ij forma o no parte de la solución óptima del problema.
- x_{ijh} representa si el arco ij es usado por el vehículo h.
- y_{ij} e y_{ijh} son las variables de flujo.
- y_{ih} indica si el vehículo h sirve al cliente i.
- c_{ij} representa el coste de transporte de la ciudad i a la ciudad j. Generalmente es función de la distancia (d_{ij}).
- f es el coste fijo por usar el vehículo. Si la flota es heterogénea, se denotará como f_h.
- d_i representa la demanda del cliente i.
- C representa la capacidad de los vehículos cuando la flota es homogénea. En caso de flota heterogénea, la capacidad de cada vehículo viene dada por C_h .
- u_i ó u_{ih}, es una variable continua que representa el espacio que le queda al vehículo después de visitar al cliente i, si se trata de un problema de recogida; o la carga que lleva el vehículo después de visitar al cliente i si

es un problema de distribución. Esta variable, al igual que ocurría en el TSP, se añade para eliminar subtours.

3.8. Consideraciones previas

1.- No está permitido ir de un punto a sí mismo, es decir, $c_{ii} = +\infty$.

2.- Cada cliente i tiene una demanda $d_i \geq 0$, y al depósito donde se encuentran los vehículos y la mercancía se le supone una demanda ficticia $d_1 = 0$.

3.- Todos los puntos de demanda pueden ser satisfechos, es decir, $d_i \leq C \; \forall \, i$.

4.- El problema es factible, es decir, $K \geq K_{min}$, con K_{min} el número mínimo de vehículos necesarios para dar servicio a todos los clientes. No siempre se trata de un dato, sino que, en ocasiones, el objetivo es minimizar los vehículos necesarios.

CAPÍTULO 4:

Clasificación de los métodos de resolución para el CVRP

Desde que se estudió por primera vez el problema de rutas de vehículos, han sido propuestos numerosos métodos para su resolución.

Dichos métodos se pueden clasificar en dos grandes grupos: algoritmos exactos y algoritmos aproximados (Figura 8). Los primeros, buscan una solución óptima, pero tienen el inconveniente de que suelen requerir tiempos de ejecución muy elevados que aumentan exponencialmente con el número de clientes a visitar, lo que dificulta enormemente su aplicación práctica. Los segundos, por su parte, dan una solución suficientemente buena (aunque no la óptima) en tiempos de ejecución menores. Para un minucioso estudio de los métodos de resolución, se puede consultar Laporte y Norbert (1987) y Toth y Vigo (1998).

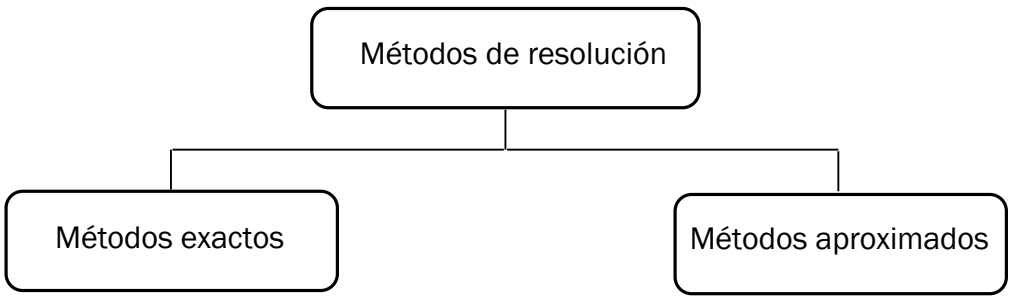

Figura 8. Clasificación de los métodos de resolución. (Elaboración propia).

4.1. Métodos exactos

4.1.1. El Algoritmo de Branch & Bound

Los métodos exactos de resolución para los problemas de rutas de vehículos suelen basarse en su formulación (a través de modelos que se verán en el Capítulo 5) como un problema de programación entera mixta (MIP, Mixed Integer Programming), es decir, con algunas variables enteras y otras binarias.

Entre los métodos exactos destacan los algoritmos de ramificación y acotación (Brach & Bound, B&B), y es precisamente éste el que utiliza el software Xpress-Mossel que se usará para la implementación de los métodos de resolución.

El paso inicial de esta técnica, consiste en resolver el problema como un problema de programación lineal, es decir, con variables no necesariamente enteras. Es lo que se conoce como problema relajado.

Una vez que se tienen las soluciones a dicho problema inicial, se comienza a hacer la ramificación de éste. Se coge una variable del problema x_j cuya solución no sea entera y se ramifica, de forma que se obtienen dos nuevos problemas, S_1 y S_2. Cada ramificación supone una restricción extra que se va añadiendo al problema (Figura 9).

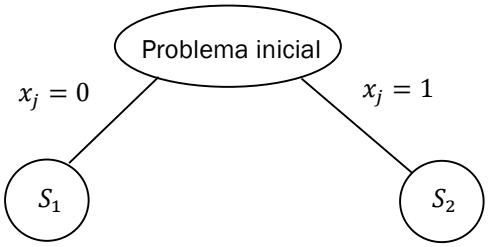

Figura 9. Primera ramificación algoritmo B&B. (Elaboración propia).

Para cada solución, se comprueba si se sigue ramificando esa rama o si por el contrario, "se poda". Para que se deje de examinar una rama, tiene que cumplirse al menos uno de los tres criterios que se muestran a continuación:

1. El problema lineal no es factible, lo que implica que el entero tampoco lo es.
2. El problema lineal da una solución entera. Se tiene por tanto, una solución factible del problema global, y se guarda dicha solución. A medida que se van obteniendo varias soluciones factibles del problema global, nos vamos quedando con la mejor de ellas.
3. La cota del problema de programación lineal es peor que el valor de la mejor solución factible.

Si no se cumplen ninguna de las tres condiciones anteriores, se vuelve a ramificar. Y así sucesivamente hasta que se tenga una única posible solución. (Figura 10).

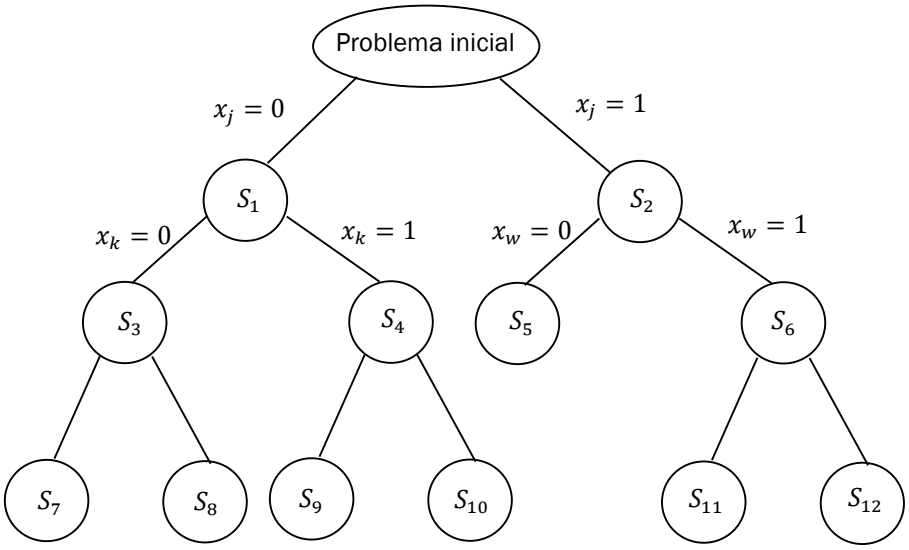

Figura 10. Ramificaciones algoritmo B&B con rama del nodo cinco podada. (Elaboración propia).

De esta manera, se puede llegar a la solución óptima sin necesidad de explorar todas y cada una de las posibles soluciones, ahorrando recursos computacionales.

4.1.2. Softwares comerciales

Para resolver problemas complejos de optimización de forma exacta, es necesario el empleo de softwares comerciales cuyo coste suele ser bastante elevado. En la actualidad, existen infinidad de éstos para la resolución de Problemas de Rutas de Vehículos. El que se ha empleado para el presente documento es Xpress- Mosel. Se trata de un software para resolver todo tipo de problemas de optimización. El problema es que su licencia es bastante cara. Para ver más, consultar http://www.fico.com/en/products/fico-xpress-optimization-suite.

4.2. Métodos aproximados

Los métodos aproximados, como ya se ha expuesto, surgen como alternativa a los métodos exactos dados los elevados tiempos necesarios para ejecutar los métodos exactos. Mediante el empleo de este tipo de técnicas, se consiguen resolver problemas con miles de ciudades en tiempos de ejecución más bajos. Como contraposición, con el empleo de heurísticas no se tiene garantía de obtener la solución óptima, aunque sí una suficientemente buena.

Existen muchos softwares comerciales para la resolución de Problemas de Rutas de Vehículos Capacitado de forma aproximada, entre los cuales se tienen:

- Axiodis: herramienta de planificación y optimización de transportes. Para ver más, consultar http://www.systeam.com.ar/.
- RoutingReparto: planificador de rutas para transporte y/o comerciales. Para ver más, consultar http://www.routingreparto.com/.
- Rutas: software desarrollado por la Universidad Politécnica de Valencia, que permite la resolución de problemas de flotas de vehículos capacitados, el cálculo de rutas y su gestión. Para ver más, consultar http://personales.upv.es/arodrigu/rutas/index.htm.

Para ver una comparativa de softwares comerciales, se puede consultar http://www.orms-today.org/surveys/Vehicle_Routing/vrss.html.

Los algoritmos aproximados que han sido propuestos para resolver Problemas de Rutas de Vehículos Capacitados pueden clasificarse en dos grandes grupos: heurísticas clásicas y metaheurísticas. El primero de ellos, a su vez, se divide en métodos constructivos, métodos de dos fases y heurísticas de mejora. (Figura 11).

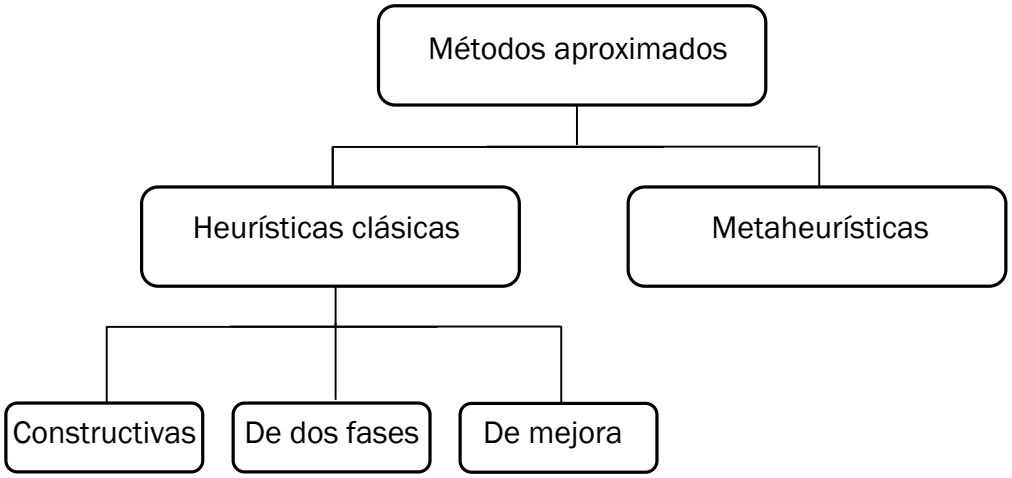

Figura 11. Clasificación métodos aproximados de resolución. (Elaboración propia).

4.2.1. Heurísticas clásicas

4.2.1.1. Heurísticas constructivas

Los procedimientos constructivos, son métodos iterativos mediante los cuales se crea una solución de manera gradual. Esto es, se parte de una solución inicial y en cada paso se añade un elemento hasta completar una solución. Se basan en seleccionar en cada iteración, el elemento con mejor evaluación.

Los métodos constructivos más usados son “El Algoritmo de Ahorros de Clarke & Wright”, las variantes de éste, y “Las heurísticas de inserción secuencial”. (Figura 12).

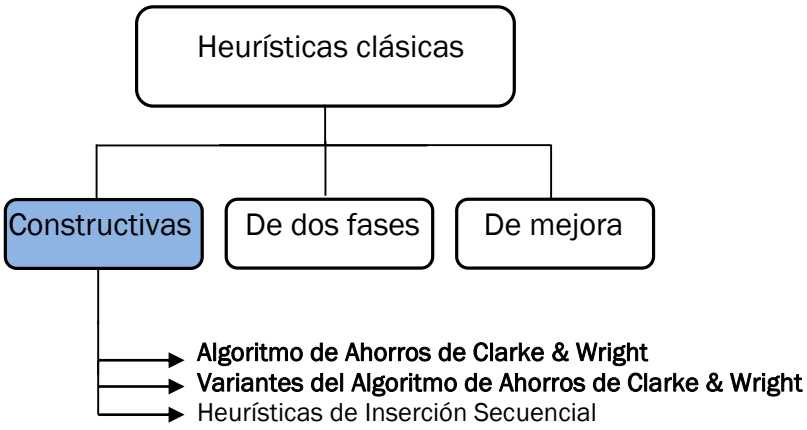

Figura 12. Clasificación heurísticas constructivas. (Elaboración propia).

Este documento se centra en el algoritmo de ahorros de Clarke & Wright original y en sus variantes, dada la importancia de este método. Para conocer el resto de heurísticas constructivas, consultar *The Vehicle Routing Problem (Paolo Toth, Daniele Vigo. Capítulo 5).*

4.2.1.2. Heurísticas de dos fases

En los procedimientos de dos fases, se descompone el problema en dos sub-problemas, agrupación de vértices en rutas factibles y construcción de la ruta, que se resuelven de manera secuencial.

Las heurísticas de dos fases se dividen en dos clases: métodos de asignar primero- rutear después (Fist cluster then routing) y métodos de rutear primero- asignar después (First routing then cluster).

Dentro del primer tipo, los métodos más usados son “El algoritmo de Barrido”, “El algoritmo de Pétalos”, “El algoritmo de Fisher y Jaikuman” y “El algoritmo de Bramel y Simchi- Levi”. (Figura 13).

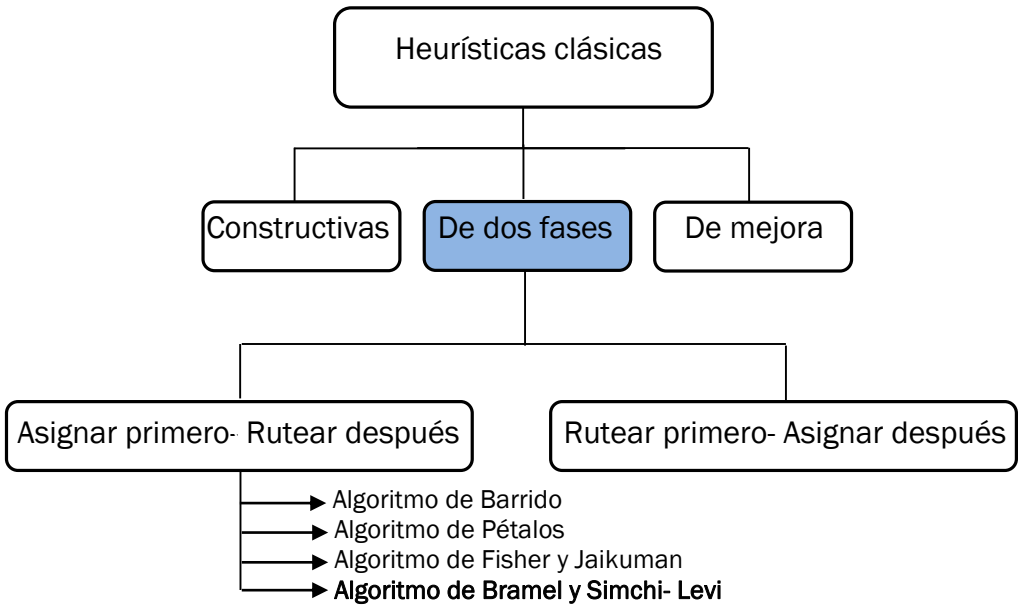

Figura 13. Clasificación heurísticas de dos fases. (Elaboración propia).

Este documento, se centra en el Algoritmo de Bramel y Simchi- Levi. Para conocer el resto de heurísticas de dos fases, consultar *The Vehicle Routing Problem (Paolo Toth, Daniele Vigo. Capítulo 5).*

4.2.1.3. Heurísticas de mejora

Los métodos de mejora intentan mejorar la calidad de cualquier solución factible realizando una serie de intercambios de vértices, ya sea dentro de una misma ruta o entre distintas rutas de vehículos.

Se trata de métodos de búsqueda local que parten de una solución ya completa y, usando el concepto de vecindario, recorren parte de espacio de búsqueda hasta encontrar un óptimo local. El vecindario de una solución s, denotado como $N(s)$, es el conjunto de soluciones se pueden construir a partir de s aplicando algún movimiento específico. Un óptimo local es una solución mejor o igual que cualquier otra solución de su vecindario. De esta forma, estos métodos, partiendo de una solución inicial, examinan su vecindario y se quedan con el mejor vecino, continuando el proceso hasta que encuentran un óptimo local. El principal inconveniente de este tipo de heurísticas es precisamente que la solución final va a depender del punto de partida.

Pueden ser exactos o heurísticos. Dentro de estos últimos, existen del tipo intra- ruta, que mueven arcos dentro de una misma ruta, entre los que se encuentran las heurísticas 2- opt y 3- opt, y extra-ruta, que los intercambian entre dos o más rutas distintas, como la heurística 2- swap. (Figura 14).

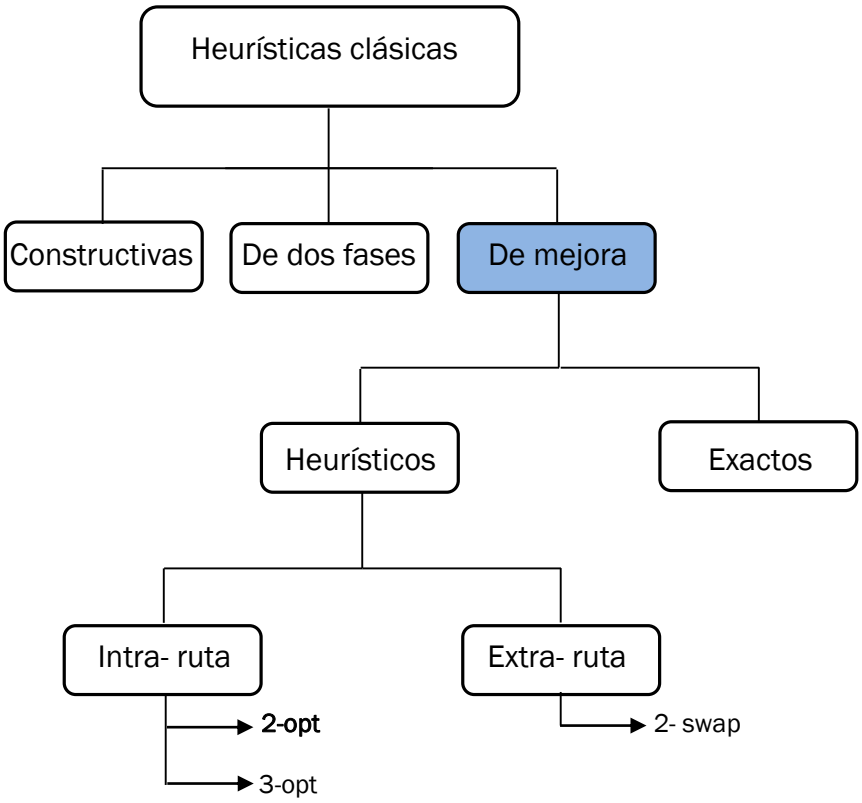

Figura 14. Clasificación heurísticas de mejora. (Elaboración propia).

Este documento, se centra en la heurística 2-opt y también se propondrá la mejora exacta del Algoritmo de Ahorros mediante la formulación de Tucker- Miller- Zemlin.

4.2.2. Metaheurísticas

En los años setenta, surgió una nueva clase de algoritmos aproximados, cuya idea básica era la de combinar diferentes métodos heurísticos para conseguir una exploración del espacio de búsqueda de forma más eficiente y efectiva. Éstas técnicas se denominan metaheurísticas.

Se pueden considerar, generalmente, seis tipos de metaheurísticas para aplicar al VRP: recocido simulado, algoritmos genéticos, redes neuronales, búsqueda tabú, colonia de hormigas y GRASP. En este documento, se explica la última de ellas. Para conocer el resto de metaheurísticas, consultar "*The Vehicle Routing Problem" (Paolo Toth, Daniele Vigo. Capítulo 6)* y *"Bio- inspired Algorithms for the Vehicle Routing Problem" (Francisco Baptista Pereira, Jorge Tavarés).*

CAPÍTULO 5:
Métodos exactos para el CVRP

Los métodos exactos de resolución se basan en la formulación de problema a través de distintos modelos y casos (flota homogénea y heterogénea).

5.1. Flota homogénea

El CVRP para flota homogénea se define de la siguiente manera: se tienen K vehículos con capacidad idéntica C, y n clientes con demandas $d_i, i = 1, \dots, n$ a distribuir o recoger. El objetivo es encontrar K trayectos que salgan del depósito, visiten una vez a cada cliente y vuelvan al punto de partida, respetando la capacidad y con coste mínimo.

5.1.1. Modelo basado en redes

$$Min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 2, \dots, n \quad (7)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 2, \dots, n \quad (8)$$

$$\sum_{i=1}^{n} x_{i1} = K \quad (9)$$

$$\sum_{i=1}^{n} x_{j1} = K \quad (10)$$

$$\sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} y_{ji} = d_i \quad i = 2, \dots, n \quad (11)$$

$$y_{ij} \leq C x_{ij} \quad i, k = 1, \dots, n \quad (12)$$

$$x_{ij} \in \{0, 1\} \quad i, j = 1, \dots, n$$

$$y_{ij} \geq 0 \quad i, j = 1, \dots, n$$

Donde:

(7) indican que de cada cliente sólo puede salir un vehículo.

(8) indican que a cada cliente sólo puede llegar un vehículo.

(9) indica que del depósito tienen que salir K vehículos.

(10) indica que al depósito tienen que llegar K vehículos.

(11) es la ecuación de Kirchoff. Indica que el flujo que sale de cada cliente menos el que entra tiene que ser igual a la demanda de dicho cliente. d_i se considera positiva si se trata de un problema de distribución y negativa si es de recogida.

(12) relaciona las dos variables del modelo y asegura que se respeta la capacidad de los vehículos.

5.1.2. Modelo de dos índices basado en Tucker-Miller-Zemlin

$$Min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 2, \dots, n \quad (7)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 2, \dots, n \quad (8)$$

$$\sum_{i=1}^{n} x_{i1} = K \quad (9)$$

$$\sum_{j=1}^{n} x_{j1} = K \quad (10)$$

$$u_i - u_j + C x_{ij} \leq C - d_j \quad i, j = 2, \dots, n \,/\, i \neq j, d_i + d_j \leq C \quad (13a)$$

$$d_i \leq u_i \leq C \quad i = 2, \dots, n \quad (13b)$$

$$x_{ij} \in \{0, 1\} \quad i, j = 1, \dots, n$$

$$u_i \geq 0 \quad i = 2, \dots, n$$

Donde:

(13) son una generalización de las restricciones de Tucker-Miller-Zemlin propuestas para el TSP.

5.1.3. Modelo de tres índices basado en Tucker-Miller-Zemlin

Surgen debido a que, en la práctica, hay ocasiones en las que se debe incluir, de manera explícita qué vehículo viaja por cada arco.

$$Min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{h=1}^{K} c_{ij}x_{ijh}$$

$$\sum_{h=1}^{K} y_{ih} = 1 \quad i = 2, \dots, n \quad (14)$$

$$\sum_{h=1}^{K} y_{1h} = K \quad (15)$$

$$\sum_{j=1}^{n} x_{ijh} = \sum_{j=1}^{n} x_{jih} = y_{ih} \quad i = 1, \dots, n; h = 1, \dots, K \quad (16)$$

$$u_{ih} - u_{jh} + Cx_{ijh} \leq C - d_j \quad i, j = 2, \dots, n, ; h = 1, \dots, K \,/$$

$$i \neq j, d_i + d_j \leq C \quad (17a)$$

$$d_i \leq u_{ih} \leq C \quad i = 2, \dots, n; h = 1, \dots, K \quad (17b)$$

$$x_{ijh} \in \{0, 1\} \quad i, j = 1, \dots, n; h = 1, \dots, K$$

$$y_{ih} \in \{0, 1\} \quad i = 1, \dots, n; h = 1, \dots, K$$

$$u_{ih} \geq 0 \quad i = 2, \dots, n; h = 1, \dots, K$$

Donde:

(14) aseguran que cada cliente es visitado una única vez.

(15) asegura que todos los vehículos llegan al punto de partida.

(16) aseguran que el mismo vehículo entra y sale de cada cliente.

(17) son una generalización de las restricciones de Tucker-Miller-Zemlin propuestas para el TSP.

5.1.4. Experiencia computacional

Si se implementan en Xpress- Mosel, los tres modelos explicados para el caso de flota homogénea (Anexo 1), los resultados obtenidos son los que se muestran en la Tabla 1.

Señalar que, se ha establecido como criterio de parada un tiempo de ejecución de 200 segundos desde que se encuentra la primera solución factible.

Para comparar dichos modelos, se ha llevado a cabo el siguiente razonamiento: para cada problema, se mirará la mejor solución factible, esto es, la menor. Si para todos los modelos es la misma, se cogerá el/los modelo/s con menor tiempo. Si por el contrario, no todos los modelos obtienen la misma solución factible, se cogerá aquel que tenga la menor siempre y cuando la diferencia de tiempo con la siguiente sea menor a sesenta segundos. De esta forma, el modelo de redes ha resultado ser el mejor en doce ocasiones, el de dos índices en once y el de tres índices en ninguna. Por tanto, el peor modelo es este último.

Es de resaltar que, el modelo de redes en tres ocasiones no es capaz de encontrar ninguna solución factible en el tiempo máximo establecido, produciéndose este hecho cuando el número de nodos es considerablemente grande.

Además, para todos los casos estudiados, a medida que aumenta el número de ciudades a recorrer, el tiempo que tarda en encontrar la solución aumenta (Gráfico 1). De hecho, a partir de treinta nodos, los tres modelos consumen el tiempo máximo establecido, lo que nos da una idea de que habrá que recurrir a métodos aproximados.

Datos	Modelo de redes (1)		Modelo de dos índices, basado en Tucker- Miller- Zemlin (2)		Modelo de tres índices, basado en Tucker- Miller- Zemlin (3)	
	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,1	290	1,9	114	0,1
eilon_13_4	290	13,8	290	1,9	290	199,3
eilon_22_4	375	3,7	375	199,812	375	200,1
eilon_23_3	569	0,9	519	0,2	569	199,3
eilon_30_3	537	199,4	520	199,7	597	199,4
eilon_31_7	1229	199,7	1048	199,7	1359	199,7
A-n32-k5	784	199,5	884	200	1055	200,3
eilon_33_4	845	199,2	886	199,8	991	200,1
A-n34-k5	782	200,3	815	199,3	974	199,9
A-n37-k6	957	199,2	1100	199,5	1127	199,6
A-n39-k6	835	199,5	921	199,5	1118	199,6
A-n44-k7	988	199,8	1031	199,3	1277	199,6
A-n46-k7	984	199,7	1060	199,9	1149	199,5
A-n48-k7	1205	199,3	1231	199,4	1483	199,4
eilon_51_5	522	200,2	715	199,377	744	200,19
A-n53-k7	1308	200,1	1103	199,8	1434	199,4
A-n55-k9	No encontrada	200,1	1145	200,2	1498	200
A-n60-k9	1753	200,1	1565	200	1888	199,3
A-n63-k10	1775	199,6	1691	199,4	1993	200,5
A-n65-k9	No encontrada	200,7	1535	199,4	1886	199,4
A-n69-k9	1444	199,3	1488	200	1811	199,6
eilon_76_7	952	200,1	752	199,519	1196	200,914
A-n80-k10	No encontrada	199,5	2351	199,9	182028	200

Tabla 1. Resultados implementación. Métodos exactos flota homogénea. (Elaboración propia).

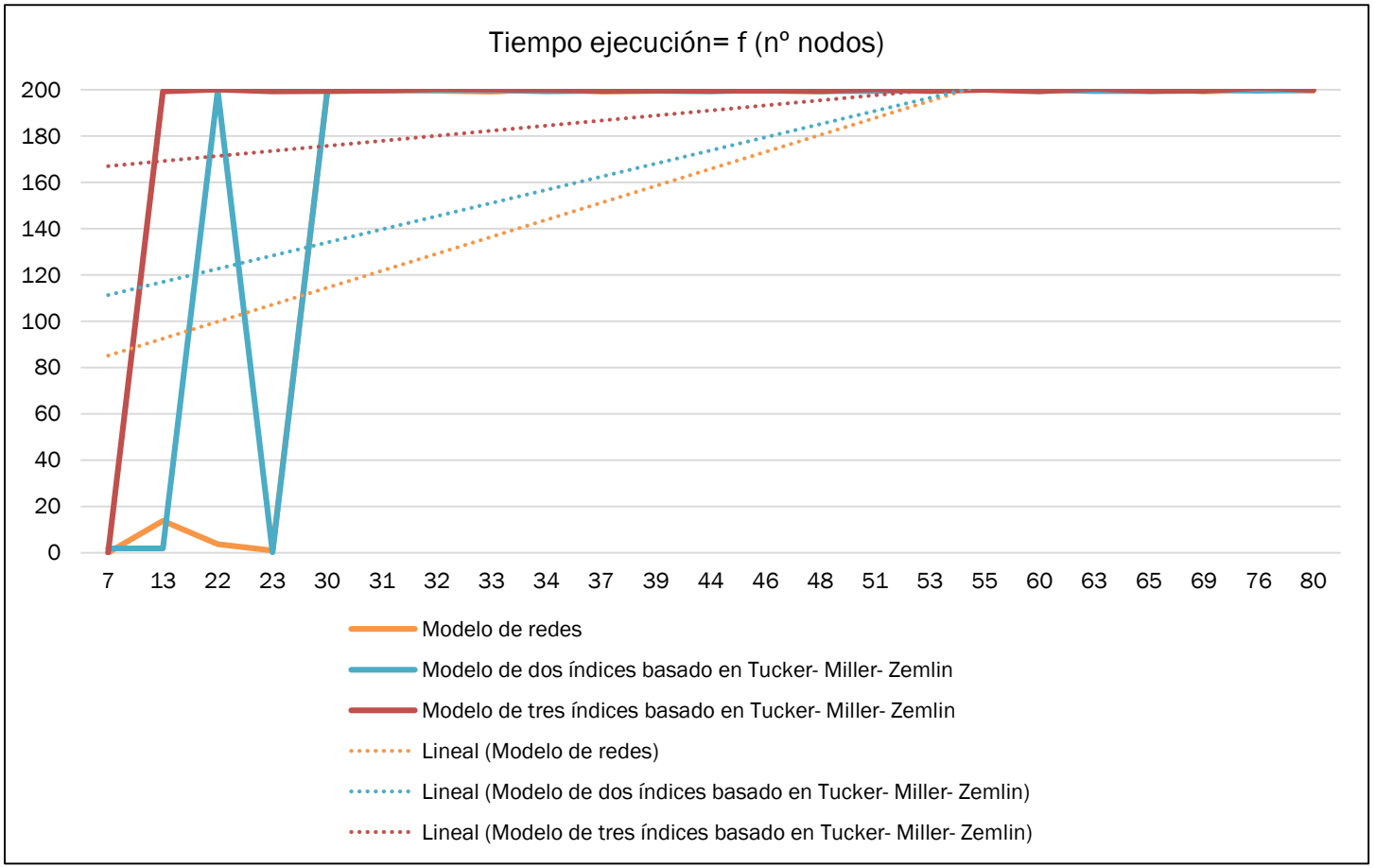

Gráfico 1. Tiempo de ejecución como función del número de nodos. Métodos exactos flota homogénea. (Elaboración propia).

5.2. Flota heterogénea

En la mayoría de aplicaciones de la vida real, no todos los vehículos tienen la misma capacidad. Es lo que se conoce como flota heterogénea.

5.2.1. Modelo de tres índices basado en Tucker- Miller- Zemlin

$$Min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{h=1}^{K} c_{ij}x_{ijh}$$

$$\sum_{h=1}^{K} y_{ih} = 1 \quad i = 2, \dots, n \quad (14)$$

$$\sum_{j=1}^{n} x_{ijh} = \sum_{j=1}^{n} x_{jih} = y_{ih} \quad i = 1, \dots, n; h = 1, \dots, K \quad (18)$$

$$u_{ih} - u_{jh} + C_h x_{ijh} \leq C_h - d_j \quad i, j = 2, \dots, n, ; h = 1, \dots, K /$$

$$i \neq j, d_i + d_j \leq C_h \quad (19a)$$

$$d_i \leq u_{ih} \leq C_h \quad i = 2, \dots, n; h = 1, \dots, K \quad (19b)$$

$$\sum_{i=1}^{n} d_i y_{ih} \leq C_h \quad (20)$$

$$x_{ijh} \in \{0, 1\} \quad i, j = 1, \dots, n; h = 1, \dots, K$$

$$y_{ih} \in \{0, 1\} \quad i = 1, \dots, n; h = 1, \dots, K$$

$$u_{ih} \geq 0 \quad i = 2, \dots, n; h = 1, \dots, K$$

Donde:

(18) aseguran que el mismo vehículo entra y sale de cada cliente.

(19) son una generalización de las restricciones de Tucker-Miller-Zemlin propuestas para el TSP.

(20) asegura que se respeta la capacidad de los vehículos.

5.2.2. Modelo de Golden- Assad- Levys- Gheysens basado en Tucker-Miller- Zemlin

$$Min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{h=1}^{k} c_{ij}x_{ijh} + \sum_{j=1}^{n}\sum_{h=1}^{K} f_h x_{ijh}$$

$$\sum_{i=1}^{n}\sum_{h=1}^{K} x_{ijh} = 1, \quad j = 2, \dots, n \quad (21)$$

$$\sum_{j=1}^{n} x_{ijh} - \sum_{j=1}^{n} x_{jih} = 0, \quad i = 1, \dots n; h = 1, \dots, K \quad (22)$$

$$u_1 = 0 \quad (23)$$

$$u_j - u_i \geq (d_j + M) \sum_{h=1}^{K} x_{ijh} - M, \quad i = 1, \dots, n; j = 2, \dots, n \quad (24)$$

$$u_j \leq \sum_{h=1}^{K} \sum_{i=1}^{n} C_h x_{ijh}, \quad j = 2, \dots, n \quad (25)$$

$$\sum_{j=2}^{n} x_{1jh} \leq 1, \quad h = 1, \dots, K \quad (26)$$

$$x_{ijh} \in \{0,1\}, \quad i, j = 1, \dots, n; h = 1, \dots, K$$

$$u_j \geq 0, \quad j = 1, \dots, n$$

Donde:

(21) indican que cada cliente es visitado una sola vez.

(22) indican que es el mismo vehículo el que entra en cada cliente que el que sale de él.

(23) indica que la demanda servida nada más salir del depósito es cero.

(24) garantizan que no se formen subtours. M es una cota superior que suele ser la suma de las capacidades.

(25) aseguran que no se supere la capacidad de los vehículos.

(26) garantizan que cada vehículo haga una sola ruta.

5.2.3. Modelo de Golden- Gheysens- Assad basado en redes

$$Min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{h=1}^{k} c_{ij} x_{ijh} + \sum_{j=1}^{n} \sum_{h=1}^{K} f_h x_{ijh}$$

$$\sum_{i=1}^{n} \sum_{h=1}^{K} x_{ijh} = 1, \quad j = 2, \dots, n \quad (21)$$

$$\sum_{j=1}^{n} x_{ijh} - \sum_{j=1}^{n} x_{jih} = 0, \quad i = 1, \dots n; h =, \dots, K \quad (22)$$

$$\sum_{j=1}^{n} y_{ij} - \sum_{j=1}^{n} y_{ji} = d_i, \quad i = 2, \dots, n \quad (11)$$

$$y_{ij} \leq \sum_{h=1}^{K} C_h \, x_{ijh} \quad i, j = 1, \dots, n \quad (27)$$

$$y_{1j} - C_h \leq M\left(1 - x_{1jh}\right) \quad j = 2, \dots, n; h = 1, \dots, K \quad (28)$$

$$\sum_{j=2}^{n} x_{1jh} \leq 1 \quad h = 1, \dots, K \quad (26)$$

$$x_{ijh} \in \{0,1\} \quad i, j = 1, \dots, n; h = 1, \dots, K$$

$$y_{ij} \geq 0 \quad i, j = 1, \dots, n$$

Donde:

(27) aseguran que se respeta la capacidad de los vehículos.

(28) relacionan las variables. M es una cota superior que suele ser $\max(Q_h) - Q_h$.

5.2.4. Modelo de Christofides- Mingozzi- Toth

$$Min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{h=1}^{k} c_{ij} x_{ijh} + \sum_{j=1}^{n} \sum_{h=1}^{K} f_h x_{ijh}$$

$$\sum_{i=1}^{n} \sum_{h=1}^{K} x_{ijh} = 1, \quad j = 2, \dots, n \quad (21)$$

$$\sum_{j=1}^{n} x_{ijh} - \sum_{j=1}^{n} x_{jih} = 0, \quad i = 1, \dots n; h =, \dots, K \quad (22)$$

$$\sum_{j=2}^{n} x_{1jh} \leq 1, \quad h = 1, \dots, K \quad (26)$$

$$\sum_{i=1}^{n} \left(d_i \sum_{j=1}^{n} x_{ijh} \right) \leq Q_h, \quad h = 1, \dots, K \quad (29)$$

$$u_i - u_j + n \sum_{h=1}^{K} x_{ijh} \leq n - 1, \quad i = 1, \dots, n; j = 2, \dots, n \quad (30)$$

Donde:

(29) son las restricciones de capacidad de los vehículos.

(30) son las restricciones de tipo Tucker-Miller-Zemlin.

5.2.5. Modelo multiproducto

$$Min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{h=1}^{k} c_{ij}x_{ijh} + \sum_{j=1}^{n}\sum_{h=1}^{K} f_h x_{ijh}$$

$$\sum_{i=1}^{n}\sum_{h=1}^{K} x_{ijh} = 1, \quad j = 2, \dots, n \quad (21)$$

$$\sum_{j=1}^{n} x_{ijh} - \sum_{j=1}^{n} x_{jih} = 0, \quad i = 2, \dots n; h =, \dots, K \quad (22)$$

$$\sum_{j=2}^{n} x_{1jh} \leq 1 \quad h = 1, \dots, K \quad (26)$$

$$y_{ijh} \leq Q_h x_{ijh}, \quad i, j = 1, \dots, n; h = 1, \dots, K \quad (31)$$

$$\sum_{j=1}^{n} y_{ijk} - \sum_{j=1}^{n} y_{jik} = -d_i \sum_{j=1}^{n} x_{1jk} \quad i = 2, \dots, n; h = 1, \dots, K \quad (32)$$

$$x_{ijh} \in \{0,1\}, \quad i, j = 1, \dots, n; h = 1, \dots, K$$

$$y_{ij} \geq 0, \quad i, j = 1, \dots, n$$

Donde:

(31) son las restricciones de capacidad de los vehículos.

(32) es la ecuación de Kirchoff.

5.2.6. Experiencia computacional

Si se implementan en Xpress- Mosel, los cinco modelos explicados para el caso de flota heterogénea (Anexo 2), los resultados obtenidos son los que se muestran en las Tablas 2 y 3.

Señalar que, se ha establecido el mismo criterio de parada que para el caso de flota homogénea.

Es de resaltar que los modelos 2 y 3 en pocas ocasiones encuentran una solución factible en el tiempo establecido. Además, en el modelo 5 se puede apreciar como cuando el número de nodos aumenta, deja de encontrar dicha solución factible, lo que nos lleva a pensar que, tal y como se viene comentando, los métodos exactos no son buenos cuando la complejidad del problema es alta.

De esta forma, descartamos los modelos 2,3 y 5 por lo que se acaba de comentar y comparamos el 1 y el 4 considerando el mismo criterio de evaluación que para los modelos de flota homogénea. Así, el mejor modelo para flota heterogénea es el modelo de tres índices basado en Tucker- Miller- Zemlin, obteniendo iguales o mejores resultados que el de Christofides- Mingozzi- Toth en el 100% de los problemas estudio, en tiempos de ejecución casi idénticos, que como se aprecia, son elevados (Gráfico 2 y Gráfico 3).

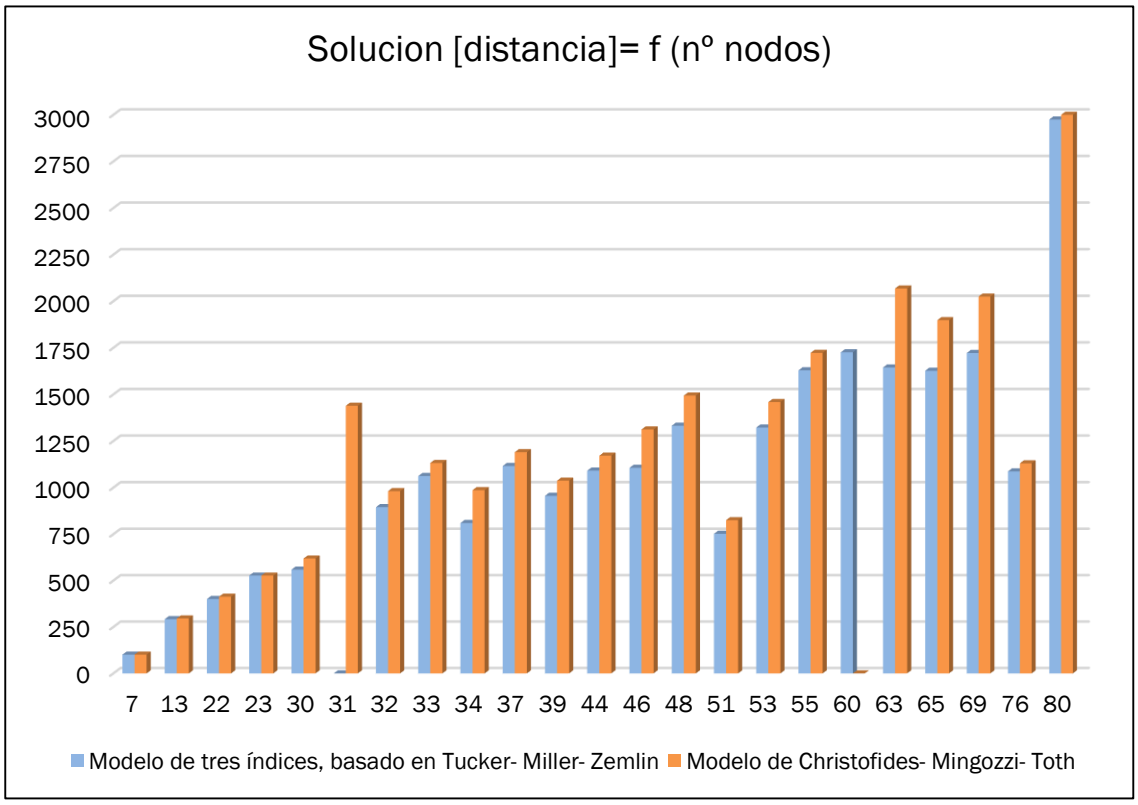

Gráfico 2. Distancia recorrida como función del número de nodos. Métodos exactos flota heterogénea. (Elaboración propia).

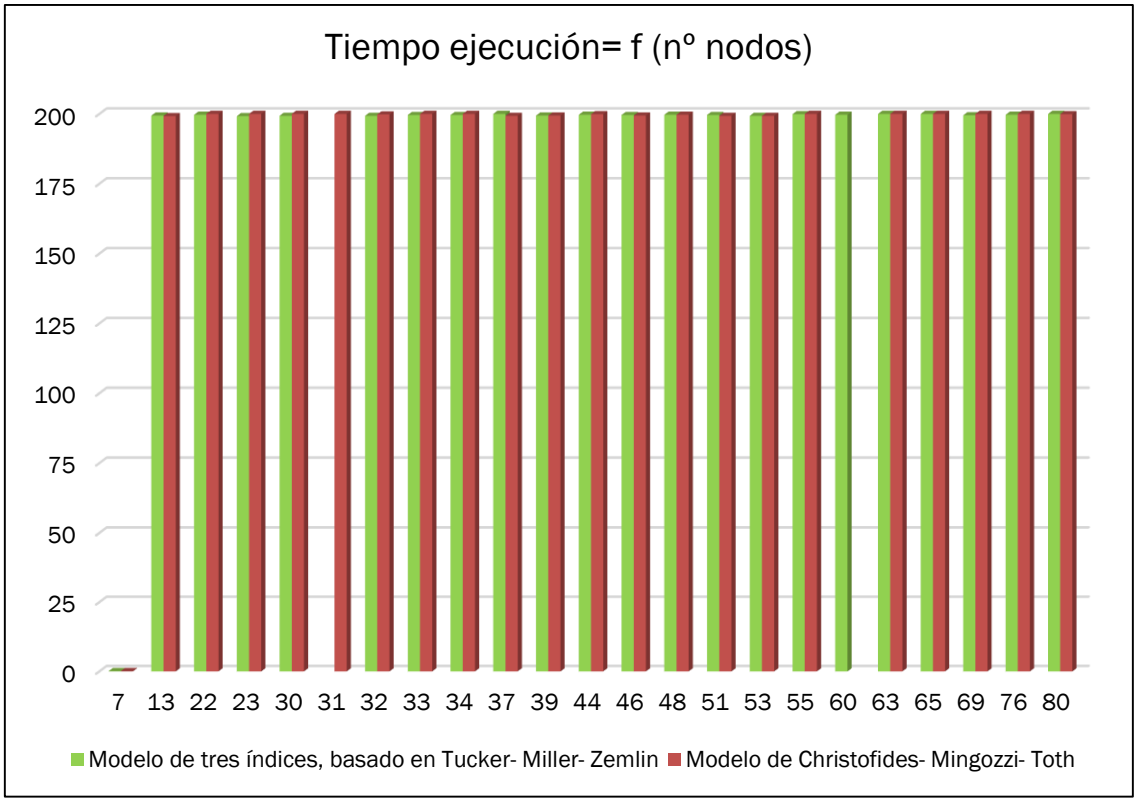

Gráfico 3. Tiempo de ejecución como función del número de nodos. Métodos exactos flota heterogénea. (Elaboración propia).

5.3. Conclusiones

Siempre se cumple que, a medida que aumenta la complejidad del problema, el tiempo que se tarda en encontrar una solución factible es cada vez más elevado, llegando incluso en bastantes ocasiones a no encontrar dicha solución.

Por lo tanto, los problemas de rutas son difícilmente optimizables en las situaciones reales por procedimientos de resolución exactos debido a este incremento del esfuerzo de cálculo necesario en relación con la dimensión del problema. Esto, como ya se comentó en el Capítulo 4, hace que se tenga que recurrir a métodos aproximados, que como se verá, proporcionan soluciones satisfactorias en tiempos de cálculo razonables, constituyendo herramientas tecnológicas capaces de incrementar la competitividad de las empresas dedicadas al transporte. A pesar de esto, dichos modelos son utilizados como base para desarrollar heurísticas y mejorar éstas, como posteriormente se verá.

Datos	Modelo de tres índices, basado en Tucker- Miller- Zemlin (1)		Modelo de Golden- Assad- Levys- Gheysens, basado en Tucker- Miller- Zemlin (2)		Modelo de Golden- Assad- Gheysens, basado en redes (3)	
	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2_heter	101	0,1	101	0,1	101	0,1
eilon_13_4_heter	292	199,4	302	199,6	292	121,4
eilon_22_4_heter	401	199,7	No encontrada		375	199,4
eilon_23_3_heter	527	199,2	527	199,3	527	2,5
eilon_30_3_heter	559	199,3	557	199,2	475	165,1
eilon_31_7_heter	No encontrada		No encontrada		No encontrada	
A-n32-k5_heter	895	199,3	1287	199,4	741	199,5
eilon_33_4_heter	1062	199,6	No encontrada		964	199,8
A-n34-k5_heter	810	199,6	No encontrada		985	199,5
A-n37-k6_heter	1116	200,2	No encontrada		No encontrada	
A-n39-k6_heter	956	199,4	No encontrada		No encontrada	
A-n44-k7_heter	1091	199,7	No encontrada		No encontrada	
A-n46-k7_heter	1106	199,6	No encontrada		No encontrada	
A-n48-k7_heter	1332	199,7	No encontrada		1282	199,7
eilon_51_5_heter	752	199,6	No encontrada		No encontrada	
A-n53-k7_heter	1322	199,3	No encontrada		No encontrada	
A-n55-k9_heter	1629	199,9	No encontrada		No encontrada	
A-n60-k9_heter	1726	199,7	No encontrada		No encontrada	
A-n63-k10_heter	1644	200	No encontrada		No encontrada	
A-n65-k9_heter	1627	200	No encontrada		No encontrada	
A-n69-k9_heter	1722	199,5	No encontrada		No encontrada	
eilon_76_7_heter	1087	199,7	No encontrada		No encontrada	
A-n80-k10_heter	2975	200,2	No encontrada		No encontrada	

Tabla 2. Resultados implementación. Métodos exactos flota heterogénea I. (Elaboración propia).

Datos	Modelo de Christofides- Mingozzi- Toth (4)		Modelo multiproducto (5)	
	Mejor solución factible (km)	Tiempo ejecución(s)	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2_heter	101	0,1	101	0,2
eilon_13_4_heter	296	199,2	292	199,6
eilon_22_4_heter	413	200,3	375	199,8
eilon_23_3_heter	527	200,3	527	21,5
eilon_30_3_heter	619	200,3	475	202,3
eilon_31_7_heter	1439	200	1278	199,6
A-n32-k5_heter	981	199,8	687	200
eilon_33_4_heter	1132	200,3	946	199,9
A-n34-k5_heter	986	200	720	199,2
A-n37-k6_heter	1190	199,3	1071	200,3
A-n39-k6_heter	1037	199,4	925	199,5
A-n44-k7_heter	1171	199,9	No encontrada	
A-n46-k7_heter	1312	199,4	No encontrada	
A-n48-k7_heter	1494	199,7	No encontrada	
eilon_51_5_heter	825	199,3	No encontrada	
A-n53-k7_heter	1459	199,3	No encontrada	
A-n55-k9_heter	1723	200,1	No encontrada	
A-n60-k9_heter	No encontrada		No encontrada	
A-n63-k10_heter	2068	200,3	No encontrada	
A-n65-k9_heter	1898	202,1	No encontrada	
A-n69-k9_heter	2025	200,2	No encontrada	
eilon_76_7_heter	1130	200,2	No encontrada	
A-n80-k10_heter	3011	199,9	No encontrada	

Tabla 3. Resultados implementación. Métodos exactos flota heterogénea II. (Elaboración propia).

CAPÍTULO 6:

Heurísticas para el CVRP

6.1. El Algoritmo de Ahorros de Clarke & Wright

6.1.1. Justificación

El Algoritmo de Ahorros de Clarke & Wirght ha sido utilizado en multitud de aplicaciones reales para resolver Problemas de Rutas de Vehículos Capacitado. En el artículo de Rand (2009) "The life and times of the Savings Method for Vehicle Routing Problems" aparecen aplicaciones reales y softwares comerciales que se han desarrollado basándose en este método. También, el artículo de Faulin (2003) "Applying MIXALG procedure in a routing problema to optimize food product delivrery", muestra su aplicación a un caso logístico real de distribución de productos vegetales en Navarra. En este, el método que se utiliza es el de Clarke & Wright más la mejora óptima con programación lineal entera, que es lo que posteriormente se planteará en el presente documento.

Esto es, aunque el método es antiguo (1964), se ha mejorado de varias formas y se sigue aplicando a casos reales y en software comercial.

6.1.2. Descripción

El Algoritmo de ahorros de Clarke & Wright es uno de los más difundidos para el VRP. Se fundamenta en el siguiente principio: si en una solución dos rutas diferentes $(0, \dots, i, 0)$ y $(0, \dots, j, 0)$ pueden ser combinadas formando una nueva ruta $(0, \dots, i, j, 0)$ como se muestra en la Figura 15 (cambios en color azul), el ahorro (en distancia) obtenido por dicha unión es:

$$s_{ij} = d_{i0} + d_{0j} - d_{ij} \quad (33)$$

pues en la nueva solución los arcos $(i, 0)$ y $(j, 0)$ no serán utilizados y se añadirá el arco (i, j).

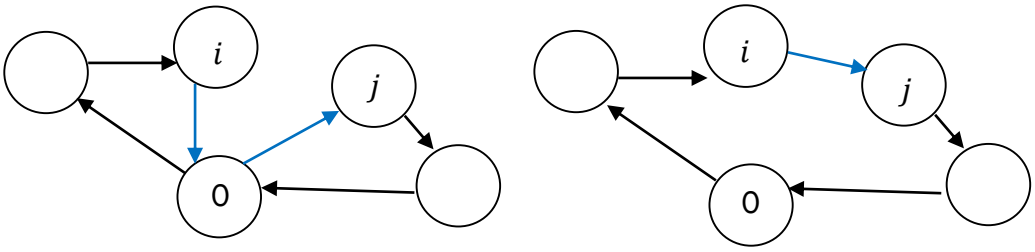

Figura 15. Dos rutas antes y después de ser unidas. Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

En este algoritmo, se parte de una solución inicial que es la unión de cada cliente con el punto 0 o depósito (Figura 16) y se realizan las uniones que den mayores ahorros, siempre y cuando se satisfagan unas restricciones que se explican en el siguiente apartado.

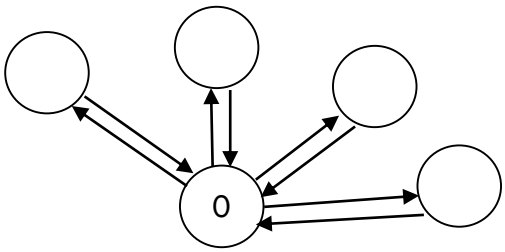

Figura 16. Solución inicial. Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Existe una versión paralela en la que se trabaja sobre todas las rutas simultáneamente, y otra secuencial que construye las rutas de una a una. Nos centraremos en la primera de ellas, pues se ha demostrado a lo largo de los años que proporciona mejores resultados.

6.1.3. Pasos a seguir

Paso 1 (cálculo de ahorros): construir la matriz de ahorros con la fórmula 33.

Paso 2 (inicialización): construir la solución inicial uniendo cada cliente con el depósito (Figura 16).

Paso 3 (mejor unión): elegir, de entre las uniones que aún no han sido evaluadas, aquella cuyo ahorro es mayor. Si la cantidad máxima de ahorro ocurre para las ciudades (i, j), éstas podrán ser enlazadas solo si se cumplen las siguientes condiciones:

a. T_{0i} y T_{0j} son mayores que 0. Éstos representan dos indicadores que tomarán el valor de 0 cuando un vehículo no atraviesa directamente entre la localidad 0 y la i y la 0 y la j, respectivamente.

b. Los clientes i y j no se encuentran en la misma ruta.
c. La unión de dichas ciudades no viola ninguna restricción, tal como la capacidad de los vehículos.

Si estas tres condiciones se cumplen, T_{ij} tomará el valor de uno.

Paso 4 (iteración): volver al Paso 3 si quedan ahorros por considerar. Sino, finalizar.

6.1.4. Ejemplo

Datos

d_{ij}	0	1	2	3	4
0	0	2	3	2	2
1		0	2	4	4
2			0	4,5	5
3				0	3
4					0

Tabla 4. Matriz de distancias. Ejemplo Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Clientes	0	1	2	3	4
Demandas	0	5	13	12	8

Tabla 5. Demandas. Ejemplo Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Capacidad vehículos	20

Tabla 6. Capacidades vehículos. Ejemplo Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Paso 1:

s_{ij}	2	3	4
1	3	0	0
2		0,5	0
3			1
3			0

Tabla 7. Matriz de ahorros. Ejemplo Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Paso 2:

Ruta 1: 0-1-0; Ruta 2: 0-2-0; Ruta 3: 0-3-0; Ruta 4: 0-4-0. (Figura 17)

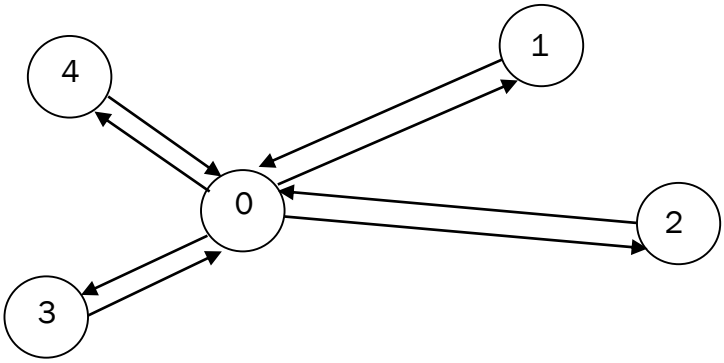

Figura 17. Solución inicial. Ejemplo Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Paso 3, primera iteración:

$\max(3; 0; 0; 0{,}5; 0; 1) = 3$ → se corresponde con los clientes 1-2.

a. $T_{01} = 1 > 0$
b. $T_{02} = 1 > 0$
c. Los clientes 1 y 2 pertenecen a rutas diferentes.
d. $dem_1 + dem_2 = 5 + 13 = 18 \leq 20$ → se cumple la restricción de capacidad.

Como se cumplen las cuatro condiciones, sí se pueden unir los clientes 1 y 2. (Figura 18), de forma que:

Ruta 1: 0-1-2-0; Ruta 2: 0-3-0; Ruta 3: 0-4-0.

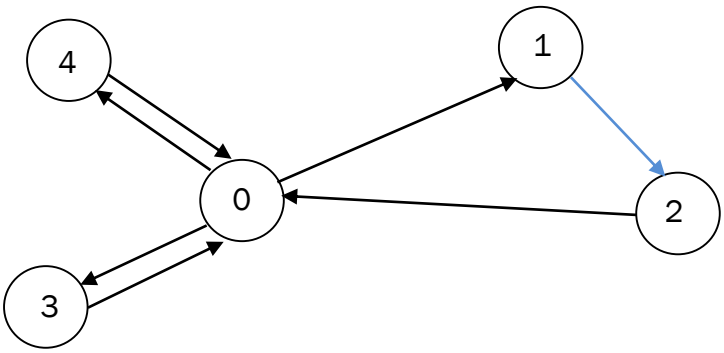

Figura 18. Primera iteración. Ejemplo Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Paso 3, segunda iteración:

$\max(0; 0; 0{,}5; 0; 1) = 1$ → se corresponde con los clientes 3-4.

a. $T_{03} = 1 > 0$
b. $T_{04} = 1 > 0$
c. Los clientes 3 y 4 pertenecen a rutas diferentes.
d. $dem_3 + dem_4 = 12 + 8 = 20 \leq 20$ → se cumple la restricción de capacidad.

Como se cumplen las cuatro condiciones, sí se pueden unir los clientes 3 y 4 (Figura 19), obteniendo:

Ruta 1: 0-1-2-0; Ruta 2: 0-3-4-0.

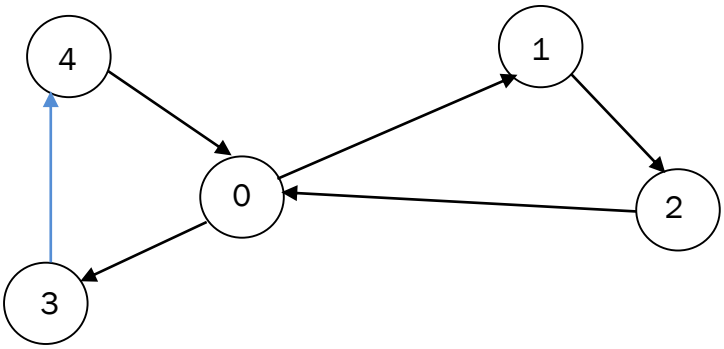

Figura 19. Segunda iteración. Ejemplo Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Paso 3, tercera iteración:

$\max(0; 0; 0{,}5; 0) = 0{,}5$ → se corresponde con los clientes 2-3.

a. $T_{02} = 1 > 0$
b. $T_{03} = 1 > 0$
c. Los clientes 2 y 3 pertenecen a rutas diferentes.
d. $dem_2 + dem_3 = 13 + 12 = 25 > 20$ → no se cumple la restricción de capacidad.

Como no se cumplen las cuatro condiciones, no se pueden unir los clientes 2 y 3.

Paso 3, cuarta iteración:

$\max(0; 0; 0) = 0$ → no se ahorra nada aunque sigamos uniendo, por lo que se deja de iterar.

Solución: por tanto, la solución mediante el Algoritmo de Ahorros de Clarke & Wright es la que obtuvimos en la segunda iteración (Figura 19).

6.1.5. Variantes del algoritmo original

Desde que Clarke & Wright propusieron el Algoritmo de Ahorros (1964) para resolver el CVRP, se han propuesto algunas variantes de la fórmula de ahorros original en forma de parametrizaciones.

Se ha observado, que utilizando la definición original del ahorro, suelen generarse algunas rutas circulares, lo cual puede ser negativo, debido a que en ocasiones puede dar lugar a la unión de puntos lejanos. (Figura 20).

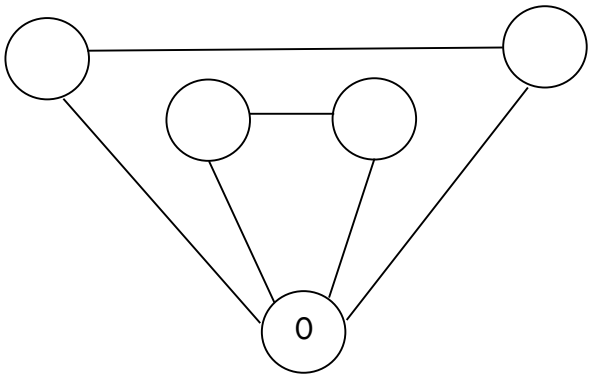

Figura 20. Formación de rutas circulares. (Elaboración propia).

Para solucionar este problema, Gaskell (1967) y Yellow (1979) introdujeron en la fórmula el parámetro $0{,}1 \leq \lambda \leq 2$, llamado "parámetro de forma de ruta". Éste penaliza la unión de rutas con clientes lejanos. Así, el cálculo de los ahorros quedaría como:

$$s_{ij} = d_{i0} + d_{0j} - \lambda d_{ij} \quad (34)$$

La fórmula propuesta hasta entonces para el cálculo de los ahorros consideraba siempre el caso simétrico. Por ello, Paessens en 1988 añadió en la expresión de ahorros un nuevo término que incluye el parámetro $0 \leq \mu \leq 2$, que es un "parámetro de peso" que tiene en cuenta la asimetría en términos de distancia desde el depósito hasta las ciudades que se están evaluando:

$$s_{ij} = d_{i0} + d_{0j} - \lambda d_{ij} - \mu\left|d_{0i} - d_{j0}\right| \quad (35)$$

Altinel y Öncan (2005) modificaron de nuevo la expresión de ahorros. La fórmula de ahorros propuesta hasta ese momento no consideraba el beneficio de usar la capacidad total de los vehículos, lo que les llevó a introducir un tercer "parámetro de peso" $0 \leq \nu \leq 2$ ligado a la idea de que cuanto más larga sea la ruta, mejor:

$$s_{ij} = d_{i0} + d_{0j} - \lambda d_{ij} - \mu\left|d_{0i} - d_{j0}\right| + \frac{\nu\left(dem_i + dem_j\right)}{demanda\ media} \qquad (36)$$

6.1.6. Experiencia computacional

Al implementar el Algoritmo de Ahorros original de Clarke & Wright (Anexo 3), se obtienen los resultados mostrados en la Tabla 8.

6.1.6.1. Comparación algoritmo original con métodos exactos

Si se comparan las soluciones obtenidas mediante el Algoritmo de Ahorros original con la mejor solución conseguida mediante los métodos exactos de resolución, se puede apreciar que, cuando el número de nodos no es muy grande, las soluciones obtenidas mediante el algoritmo de ahorros son algo peores que las de los métodos exactos aunque como contraposición estos últimos emplean mayor tiempo computacional. Sin embargo, cuando el número de nodos es considerablemente grande, el algoritmo de ahorros proporciona mejores soluciones en tiempo mucho menores. (Gráfico 4 y Gráfico 5).

Datos	Algoritmo de Ahorros de Clarke & Wright	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	119	0,016
eilon_13_4	290	0
eilon_22_4	387	0,031
eilon_23_3	577	0,016
eilon_30_3	532	0,047
eilon_31_7	1263	0,078
A-n32-k5	832	0,236
eilon_33_4	843	0,438
A-n34-k5	810	0,273
A-n37-k6	981	0,391
A-n39-k6	848	0,472
A-n44-k7	1010	0,765
A-n46-k7	940	0,91
A-n48-k7	1112	1,087
eilon_51_5	582	0,547
A-n53-k7	1135	1,582
A-n55-k9	1111	1,784
A-n60-k9	1367	2,626
A-n63-k10	1353	3,146
A-n65-k9	1263	3,407
A-n69-k9	1210	4,288
eilon_76_7	747	2,152
A-n80-k10	1818	8,157

Tabla 8. Resultados implementación. Algoritmo de Ahorros original de Clarke & Wright. (Elaboración propia).

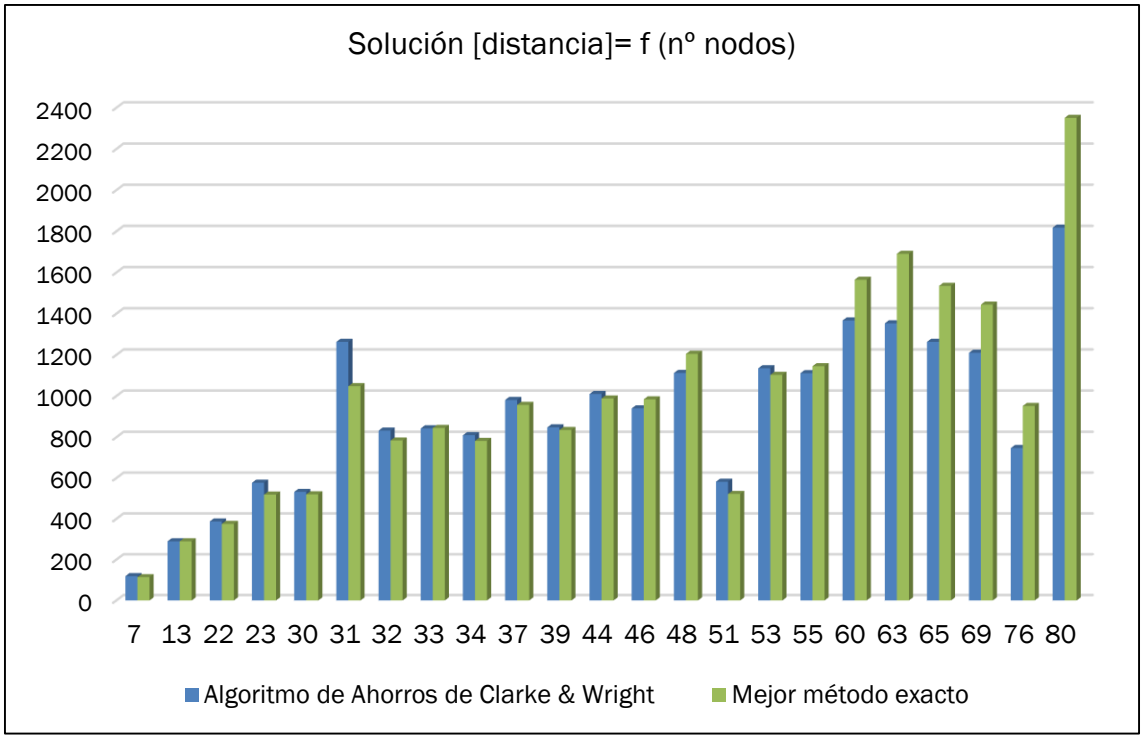

Gráfico 4. Distancia recorrida como función del número de nodos. Comparación Algoritmo Ahorros original con mejor método exacto. (Elaboración propia).

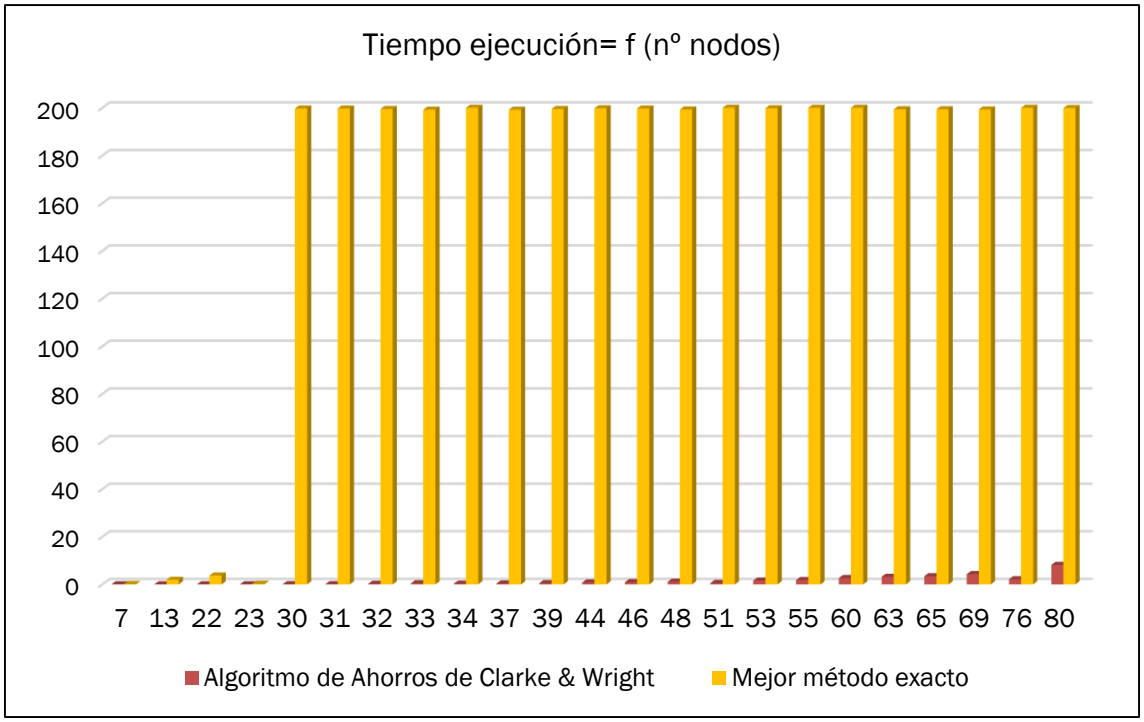

Gráfico 5. Tiempo de ejecución como función del número de nodos. Comparación Algoritmo Ahorros original con mejor método exacto. (Elaboración propia).

6.1.6.2. Comparación algoritmo original con sus variantes

Al implementar las tres variantes a la fórmula original del ahorro, se obtienen los resultados que figuran en la Tabla 9 y Tabla 10. Señalar que:

- Para la modificación de un parámetro, se ha probado con 20 valores de lambda, comprendidos entre 0,1 y 2.
- Para la modificación de dos parámetros, se ha probado con 10 valores de lambda comprendidos entre 0,4 y 1,8 y 8 valores de mu entre 0,6 y 2.
- Para la modificación de tres parámetros, se ha probado con 10 valores de lambda, 8 valores de mu y 8 de fi, comprendidos entre 0,4-1,8, 0,6-2 y 0,6-1,8, respectivamente.

Comparándolo con el algoritmo original, para el caso de la modificación de uno y dos parámetros, en el 70% de los casos se obtienen mejores soluciones mediante la modificación (Gráfico 6 y Gráfico 7, respectivamente), pero mediante tiempos de resolución bastante más grandes que aumentan exponencialmente con la complejidad (Gráfico 8 y Gráfico 9, respectivamente).

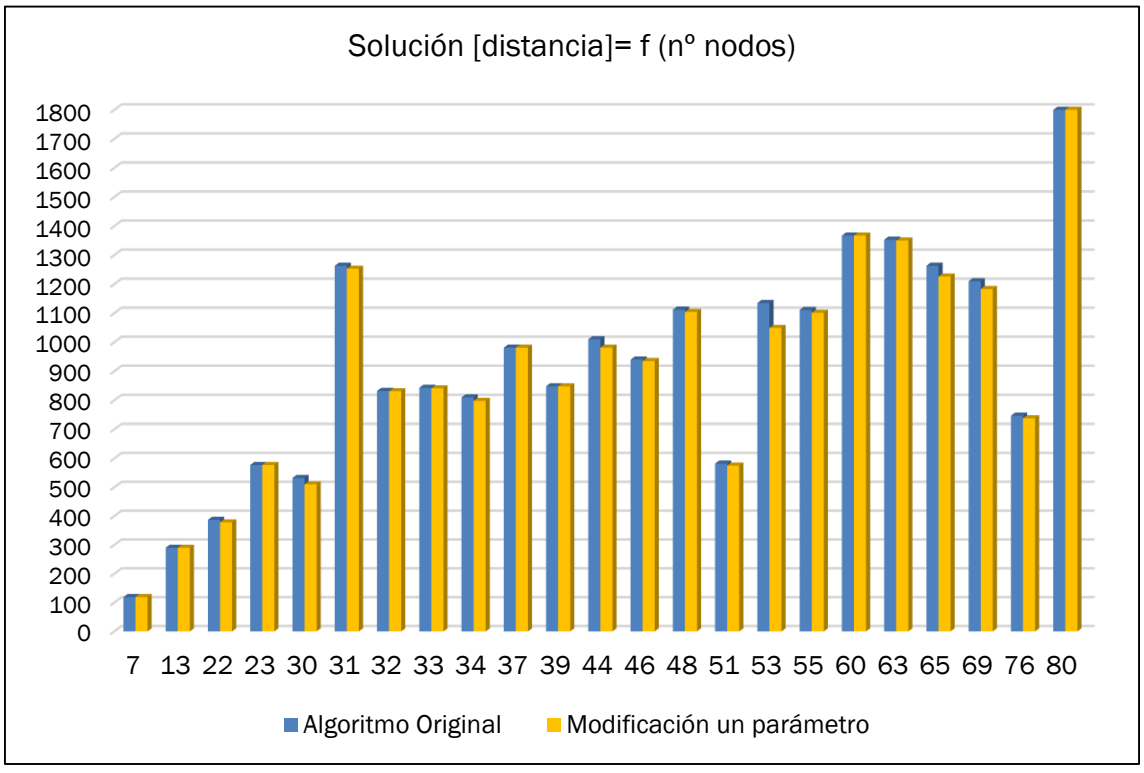

Gráfico 6. Distancia recorrida como función del número de nodos. Comparación Algoritmo Ahorros original con modificación un parámetro. (Elaboración propia).

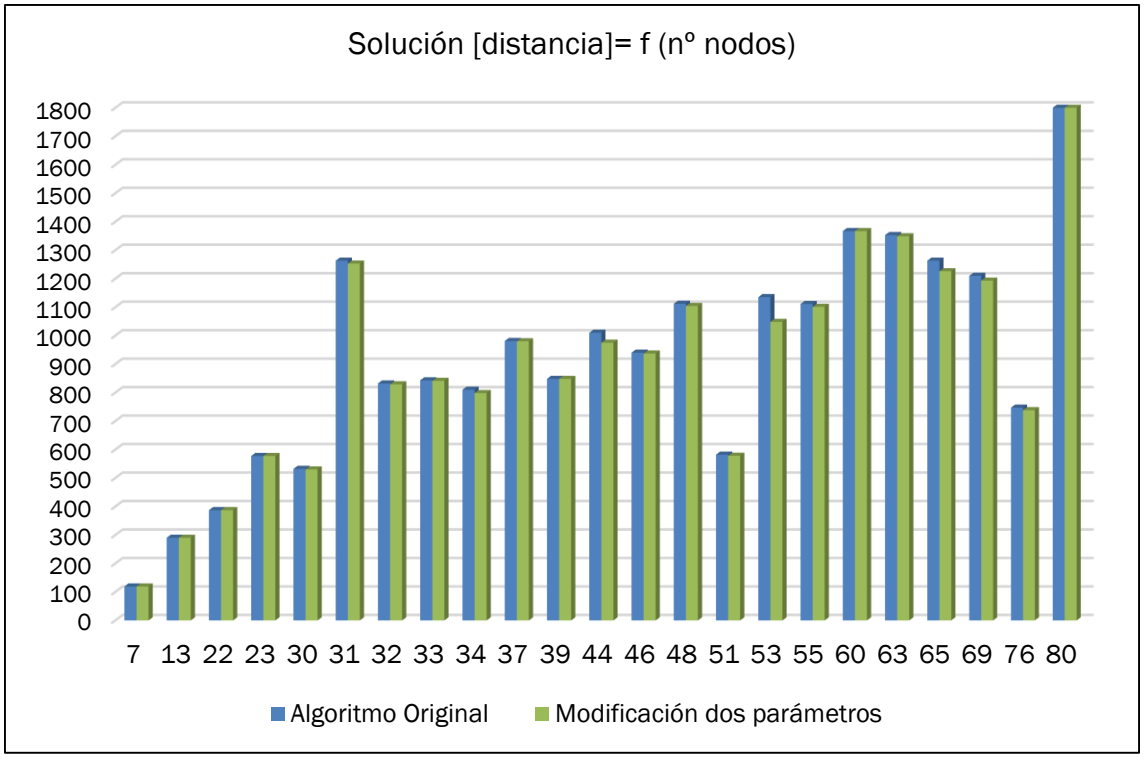

Gráfico 7. Distancia recorrida como función del número de nodos. Comparación Algoritmo Ahorros original con modificación dos parámetros. (Elaboración propia).

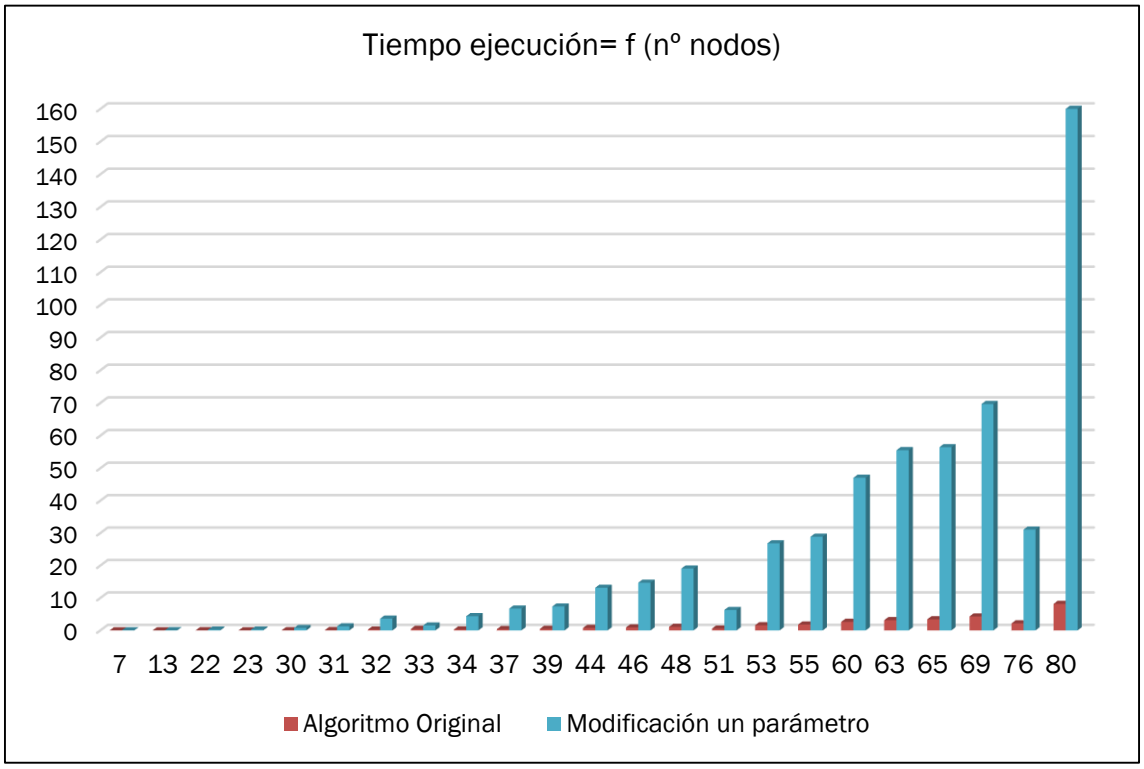

Gráfico 8. Tiempo de ejecución como función del número de nodos. Comparación Algoritmo Ahorros original con modificación un parámetro. (Elaboración propia).

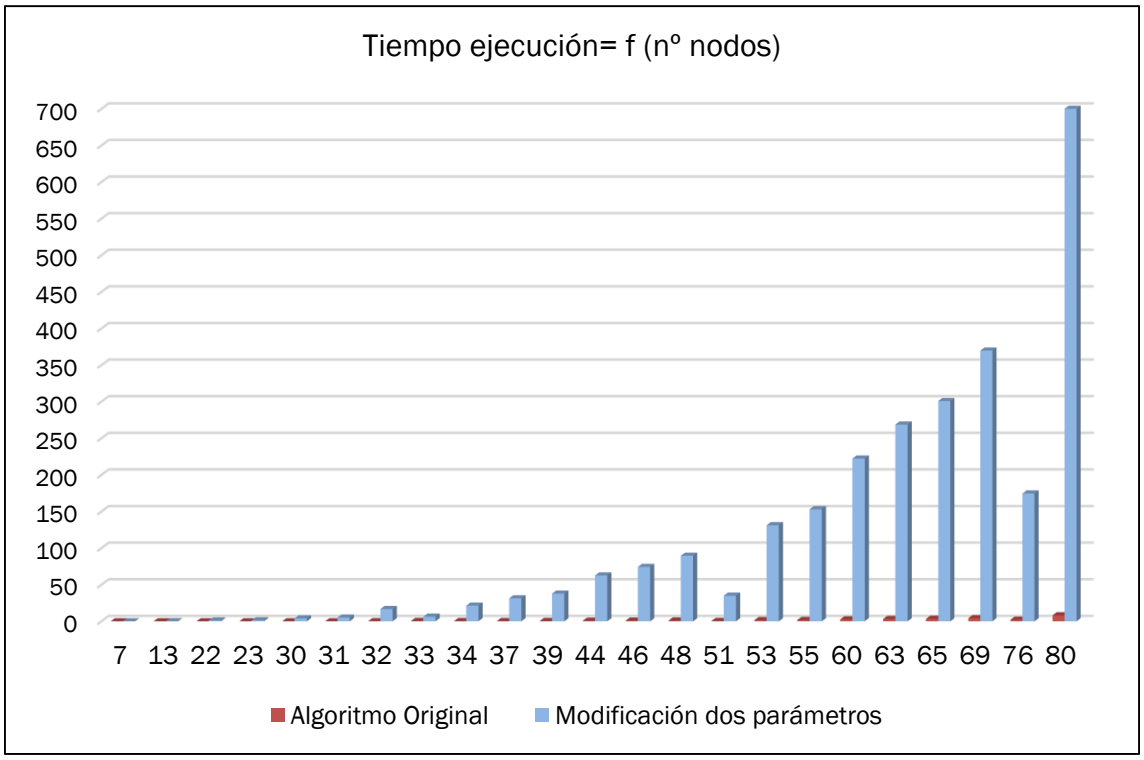

Gráfico 9. Tiempo de ejecución como función del número de nodos. Comparación Algoritmo Ahorros original con modificación dos parámetros. (Elaboración propia).

Además, si nos fijamos en la modificación de uno y dos parámetros, las soluciones que se consiguen son muy similares en ambos (Gráfico 10), pero el de dos parámetros emplea unos tiempos de ejecución mayores, que se ve acentuado con el aumento de la complejidad del problema (Gráfico 11), de forma que no es aconsejable emplear recursos computacionales en este último.

Con respecto a la modificación de tres parámetros, no merece la pena su implementación, pues en la mayoría de las ocasiones necesita más de 500 segundos para encontrar una solución. (Tabla 10).

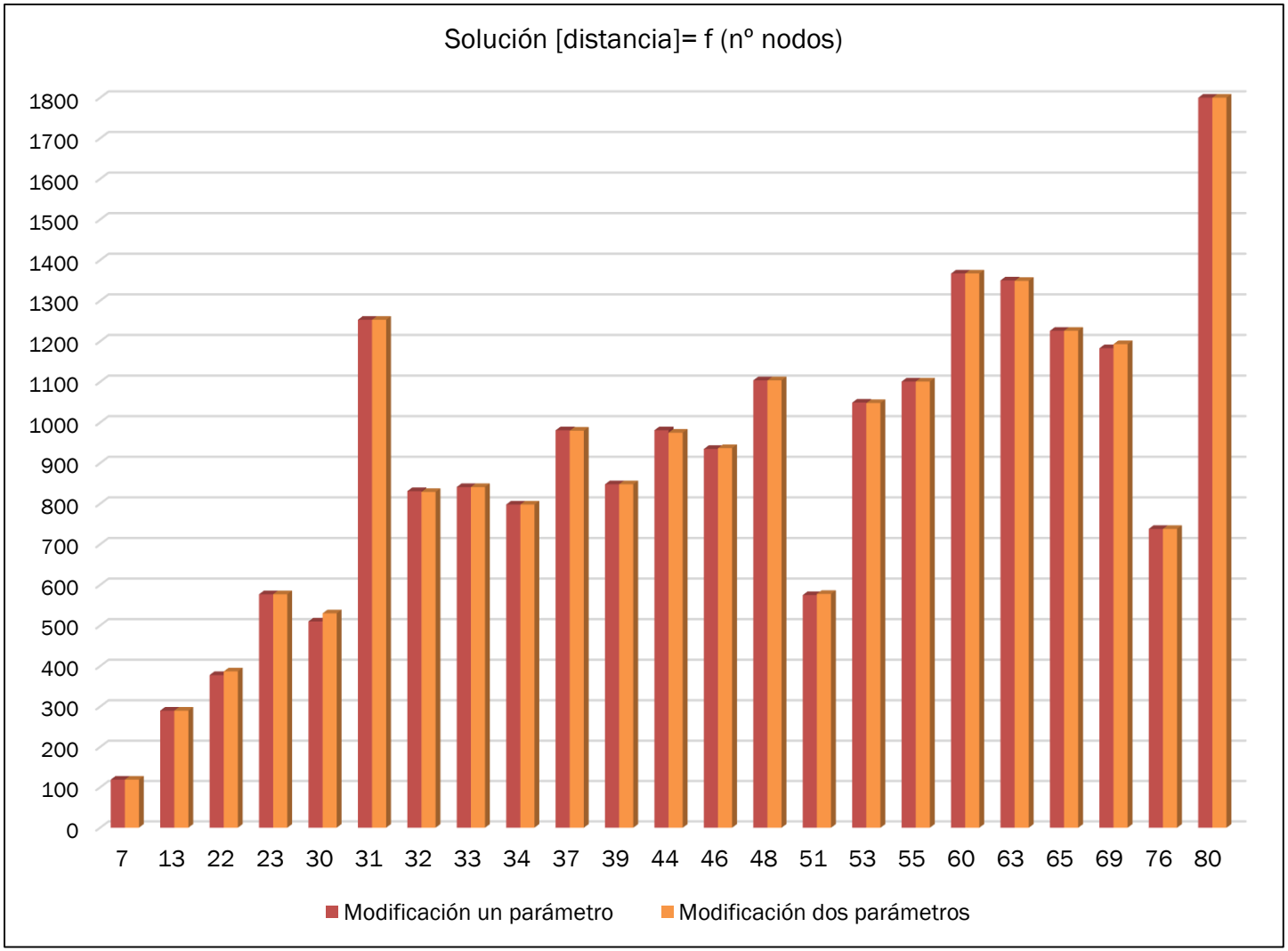

Gráfico 10. Distancia recorrida como función del número de nodos. Comparación modificación uno y dos parámetros. (Elaboración propia)

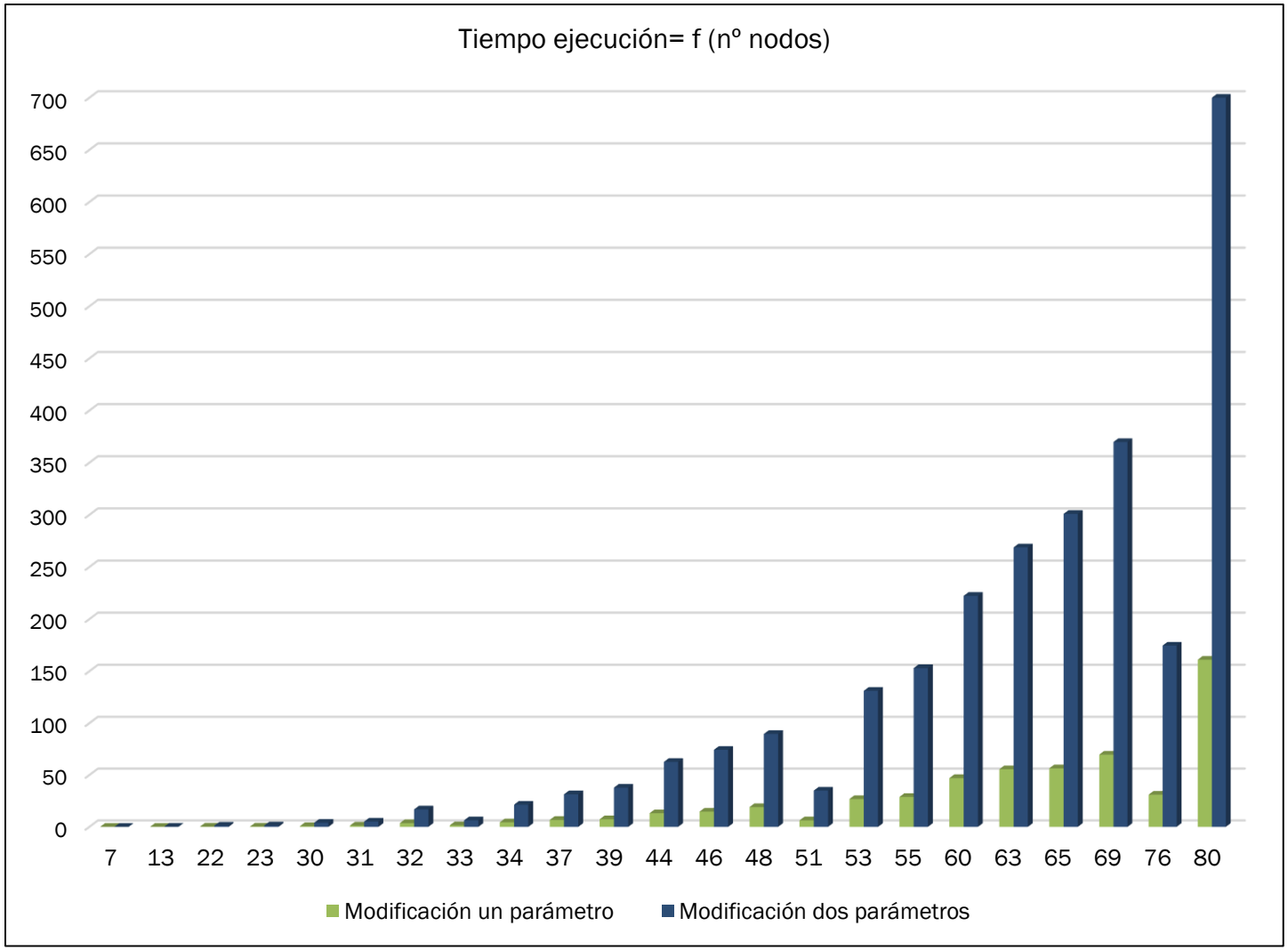

Gráfico 11. Tiempo de ejecución como función del número de nodos. Comparación modificación uno y dos parámetros. (Elaboración propia).

Datos	Modificación un parámetro			Modificación dos parámetros			
	Lambda	Mejor solución factible (km)	Tiempo ejecución (s)	Lambda	Mu	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	0,1	119	0,01	0,4	0,6	119	0,016
eilon_13_4	0,9	290	0,031	1,33333	0,6	290	0,156
eilon_22_4	1,3	378	0,218	1,17778	0,6	387	1,171
eilon_23_3	0,6	577	0,25	0,866667	0,6	577	1,342
eilon_30_3	1,7	510	0,733	1,48889	0,6	530	3,947
eilon_31_7	1,1	1253	1,233	1,8	0,6	1253	5,101
A-n32-k5	1,2	831	3,619	0,4	1,4	829	16,821
eilon_33_4	1,1	841	1,498	1,64444	0,6	841	6,35
A-n34-k5	0,7	798	4,371	1,17778	0,6	798	21,343
A-n37-k6	1	981	6,707	1,48889	0,6	980	31,384
A-n39-k6	1	848	7,347	1,48889	0,6	848	37,805
A-n44-k7	0,9	981	13,123	1,33333	0,6	975	62,628
A-n46-k7	1,2	935	14,69	1,33333	0,6	937	74,287
A-n48-k7	1,1	1104	19,056	1,64444	0,6	1104	89,605
eilon_51_5	0,5	575	6,288	1,64444	0,6	578	34,972
A-n53-k7	0,6	1049	26,833	1,33333	0,6	1048	131,198
A-n55-k9	1,1	1101	28,859	1,64444	0,6	1101	153,037
A-n60-k9	1	1367	47,039	1,64444	0,6	1367	222,297
A-n63-k10	1,1	1350	55,534	1,64444	0,6	1349	268,732
A-n65-k9	0,9	1226	56,438	1,33333	0,6	1226	300,882
A-n69-k9	1,3	1183	69,658	1,8	0,6	1193	369,82
eilon_76_7	1,1	738	31,064	1,64444	0,6	738	174,57
A-n80-k10	1	1818	161,065	1,8	0,8	1818	718,091

Tabla 9. Resultados implementación. Modificación uno y dos parámetros del Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Datos	Modificación tres parámetros				
	Lambda	Mu	Fi	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	0,4	0,6	0,6	119	0,078
eilon_13_4	1,33333	0,6	0,6	290	1,186
eilon_22_4	1,02222	0,6	0,6	387	9,362
eilon_23_3	0,866667	0,6	0,6	577	10,765
eilon_30_3	1,8	0,6	1,8	506	31,813
eilon_31_7	1,8	0,6	1,28571	1252	41,006
A-n32-k5	0,4	1,6	1,45714	827	136,58
eilon_33_4	1,48889	0,6	0,6	841	52,208
A-n34-k5	1,64444	0,6	0,6	793	172,337
A-n37-k6	1,48889	0,6	0,6	980	251,726
A-n39-k6	1,64444	0,6	0,6	844	303,932
A-n44-k7					>500
A-n46-k7					>500
A-n48-k7					>500
eilon_51_5	1,17778	1,8	0,6	575	283,645
A-n53-k7					>500
A-n55-k9					>500
A-n60-k9					>500
A-n63-k10					>500
A-n65-k9					>500
A-n69-k9					>500
eilon_76_7	1,8	1	1,8	703	>500
A-n80-k10					>500

Tabla 10. Resultados implementación. Modificación tres parámetros del Algoritmo de Ahorros de Clarke & Wright. (Elaboración propia).

Si se recopilan los valores de lambda en el caso de la modificación de un parámetro, el valor que más se repite es 1,1 (26 % de las veces), estando entre 0,9 y 1,3 el 74 % de las ocasiones (Tabla 11).

Lambda	Nº veces
0,1	1
0,5	1
0,6	2
0,7	1
0,9	3
1	4
1,1	6
1,2	2
1,3	2
1,7	1

Tabla 11. Recopilación valores de lambda para modificación un parámetro. (Elaboración propia).

Si se hace lo mismo para la modificación de dos parámetros, el valor de landa que más se repite es 1,64444 (30 % de las veces), estando entre 1,3333 y 1,8 el 78% de ellas (Tabla 12), mientras que el valor de mu que más veces está presente es 0,6 (90 % de las veces. Tabla 13). Además, la combinación más repetida es lambda 1,64444 y mu 0,6 (30 % de las ocasiones. Tabla 14).

Lambda	Nº veces
0,4	2
0,866667	1
1,17778	2
1,3333	5
1,48889	3
1,64444	7
1,8	3

Tabla 12. Recopilación valores de lambda para modificación dos parámetros. (Elaboración propia).

Mu	Nº veces
0,6	21
0,8	1
1,4	1

Tabla 13. Recopilación valores de mu para modificación dos parámetros. (Elaboración propia).

Lambda	Nu	Nº veces
0,4	0,6	1
0,4	1,4	1
0,866667	0,6	1
1,17778	0,6	2
1,3333	0,6	5
1,48889	0,6	3
1,64444	0,6	7
1,8	0,6	2
1,8	0,8	1

Tabla 14. Recopilación combinación valores de lambda y mu para modificación dos parámetros. (Elaboración propia).

6.2. Heurística de Localización de Bramel y Simchi- Levi

6.2.1. Introducción

Los métodos de asignar primero y rutear después se llevan a cabo en dos fases. En la primera, se busca generar grupos de clientes, llamados clusters, que estarán en la misma ruta en la solución final. La segunda, consiste en crear para cada cluster, una ruta que visite a todos sus clientes, es decir, resolver para cada cluster, un TSP. Las restricciones de capacidad son consideradas en la primera de las etapas, asegurando que la demanda total de cada cluster no supera la capacidad de los vehículos.

6.2.2. Descripción

Primera fase: creación de clusters.

Los clusters son determinados resolviendo un Problema de Localización de Concentradores con Capacidades (CCLP). Dicho problema se define como: "se dispone de m posibles ubicaciones para concentradores de capacidad Q_j $(j = 1, \dots, m)$ y n ciudades con demanda d_i $(i = 1, \dots, n)$. El coste por colocar un concentrador en la ubicación j es f_j y el coste de conectar la ciudad i con el concentrador j es c_{ij}". El CCLP consiste en decidir cuántos concentradores colocar y qué ciudades conectar a cada concentrador de modo que cada ciudad se conecte con un solo concentrador, se satisfagan las restricciones de capacidad y se minimicen los costes. La formulación de dicho problema es la siguiente:

$$Min \sum_{j=1}^{m} f_j y_j + \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\sum_{i=1}^{n} d_i x_{ij} \leq Q_j \quad j = 1, \dots, m \quad (37)$$

$$x_{ij} \leq y_j \quad i = 1, \dots, n; j = 1, \dots, m \quad (38)$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad i = 1, \dots, n \quad (39)$$

$$\sum_{j=1}^{m} y_j = m \quad (40)$$

$$x_{ij} \in \{0,1\} \quad i = 1, \dots, n; j = 1, \dots, m$$

$$y_j \in \{0,1\} \quad j = 1, \dots, m$$

Donde:

x_{ij} indica si la ciudad i se conecta al concentrador j.

y_j indica si se instala un concentrador en el sitio j.

$$c_{ij} = d_{1i} + d_{ij} - d_{1j} \quad (41)$$

$$f_j = 2d_{1j} \quad (42)$$

(37) aseguran que se respetan las capacidades.

(38) relaciona las dos variables de decisión.

(39) aseguran que cada ciudad se conecta exactamente a un concentrador.

(40) asegura que el número de concentradores que se instalan es m.

Segunda fase: resolución de TSP.

Para cada cluster, se resuelve un TSP, ya sea mediante el modelo de redes o el de Tucker-Miller-Zemlin (Capítulo 2).

6.2.3. Experiencia computacional

Al implementar en Xpress- Mossel la Heurística de Localización de Bramel y Simchi- Levi (Anexo 3), se obtienen los resultados que se muestran en la Tabla 15.

Datos	Heurística de Localización de Bramel y Simchi-Levi	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,031
eilon_13_4	290	0,156
eilon_22_4	383	0,063
eilon_23_3	610	0,156
eilon_30_3	541	11,623
eilon_31_7	1212	5,725
A-n32-k5	830	0,858
eilon_33_4	843	1,513
A-n34-k5	830	1,388
A-n37-k6	971	3,417
A-n39-k6	851	0,546
A-n44-k7	986	1,685
A-n46-k7	917	0,764
A-n48-k7	1073	0,796
eilon_51_5	535	0,64
A-n53-k7	1047	20,055
A-n55-k9	1087	8,197
A-n60-k9	1364	199,845
A-n63-k10	1337	83,642
A-n65-k9	1200	12,091
A-n69-k9	1191	5,304
eilon_76_7	701	3,105
A-n80-k10	1788	244,371

Tabla 15. Resultados implementación. Heurística de Localización de Bramel y Simchi- Levi. (Elaboración propia).

6.2.3.1. Comparación con métodos exactos

Al igual que ocurría con el Algoritmo de Ahorros, cuando el número de nodos no es muy grande, las soluciones obtenidas con la heurística de localización son algo peores que las de los métodos exactos pero emplean menor tiempo computacional. A medida que el número de nodos aumenta, la heurística va proporcionando mejores resultados que el método exacto el tiempos computacionales mucho menores (salvo en dos casos que emplea el mismo tiempo). (Gráfico 12 y Gráfico 13).

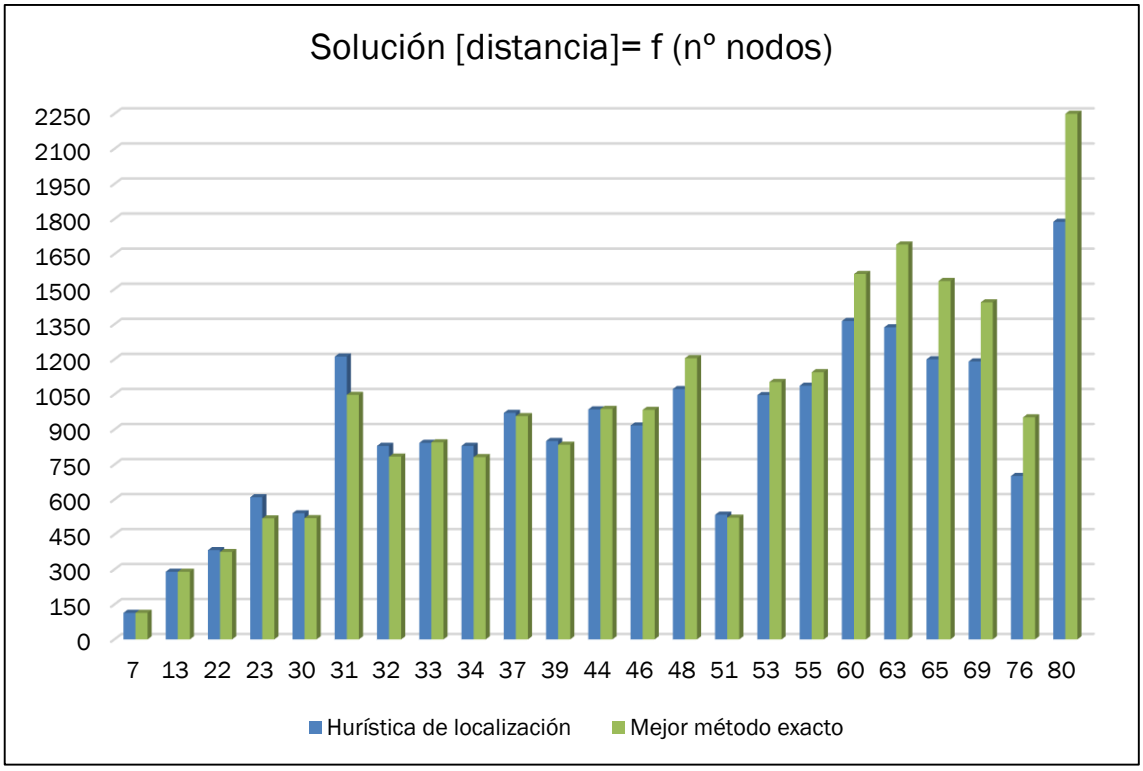

Gráfico 12. Distancia recorrida como función del número de nodos. Comparación Heurística de Localización de Bramel y Simchi- Levi con mejor método exacto. (Elaboración propia).

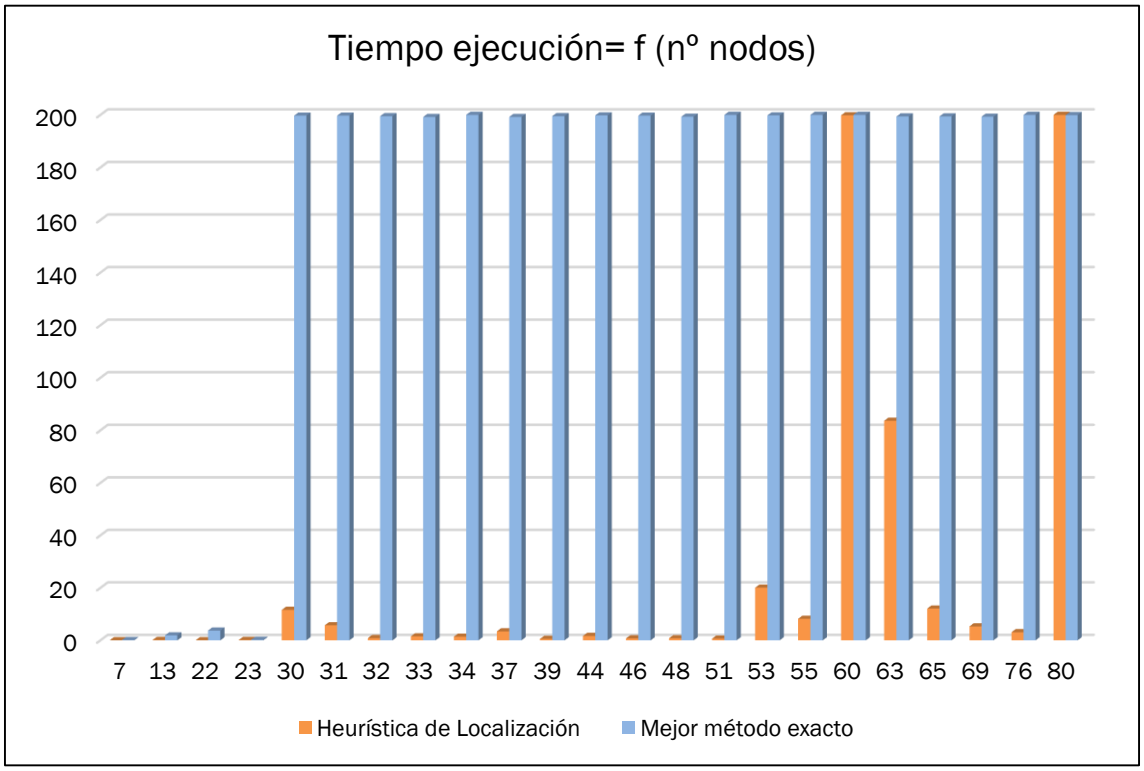

Gráfico 13. Tiempo de ejecución como función del número de nodos. Comparación Heurística de Localización de Bramel y Simchi- Levi con mejor método exacto. (Elaboración propia).

6.2.3.2. Comparación con Algoritmo de Ahorros original

La Heurística de Localización es mejor que el Algoritmo de Ahorros en el 74 % de los casos, hecho que se produce sobre todo a medida que el número de nodos es mayor. (Gráfico 14).

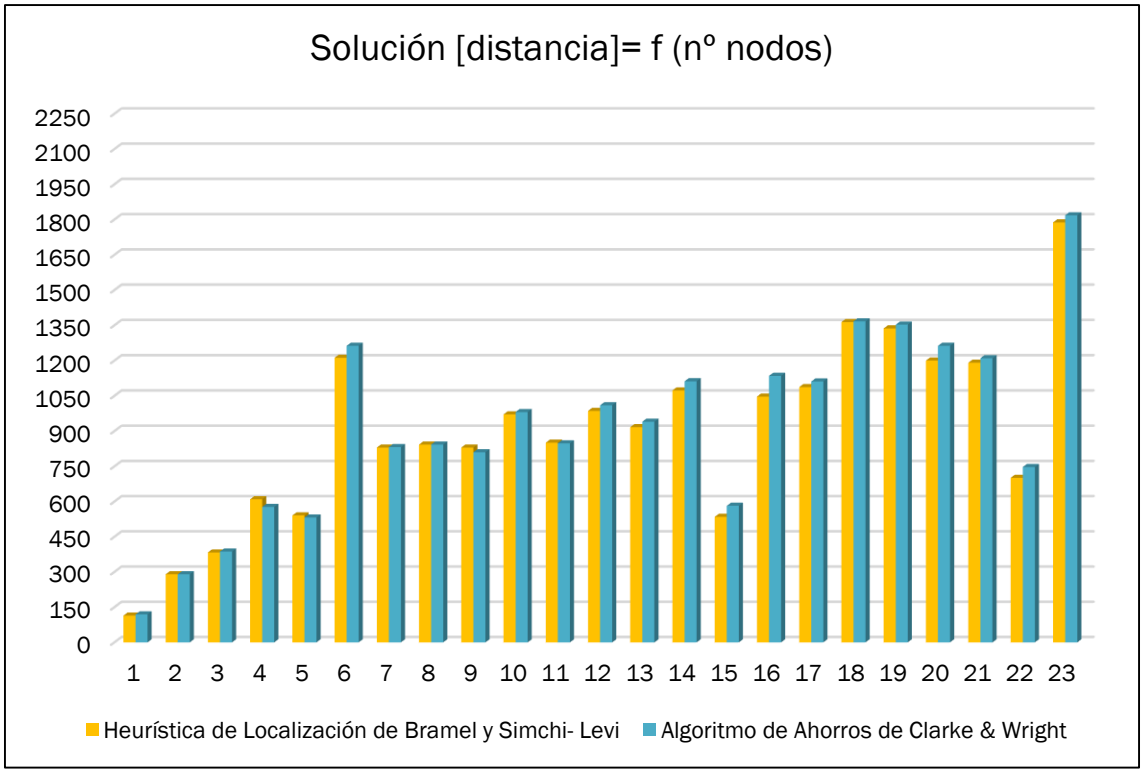

Gráfico 14. Distancia recorrida como función del número de nodos. Comparación Heurística de Localización de Bramel y Simchi- Levi con Algoritmo de Ahorros original. (Elaboración propia).

Además, la máxima diferencia de tiempo entre ambos métodos aproximados es de nueve segundos (Gráfico 15), por lo que podemos concluir que la Heurística del Concentrador es mejor que el Algoritmo de Ahorros.

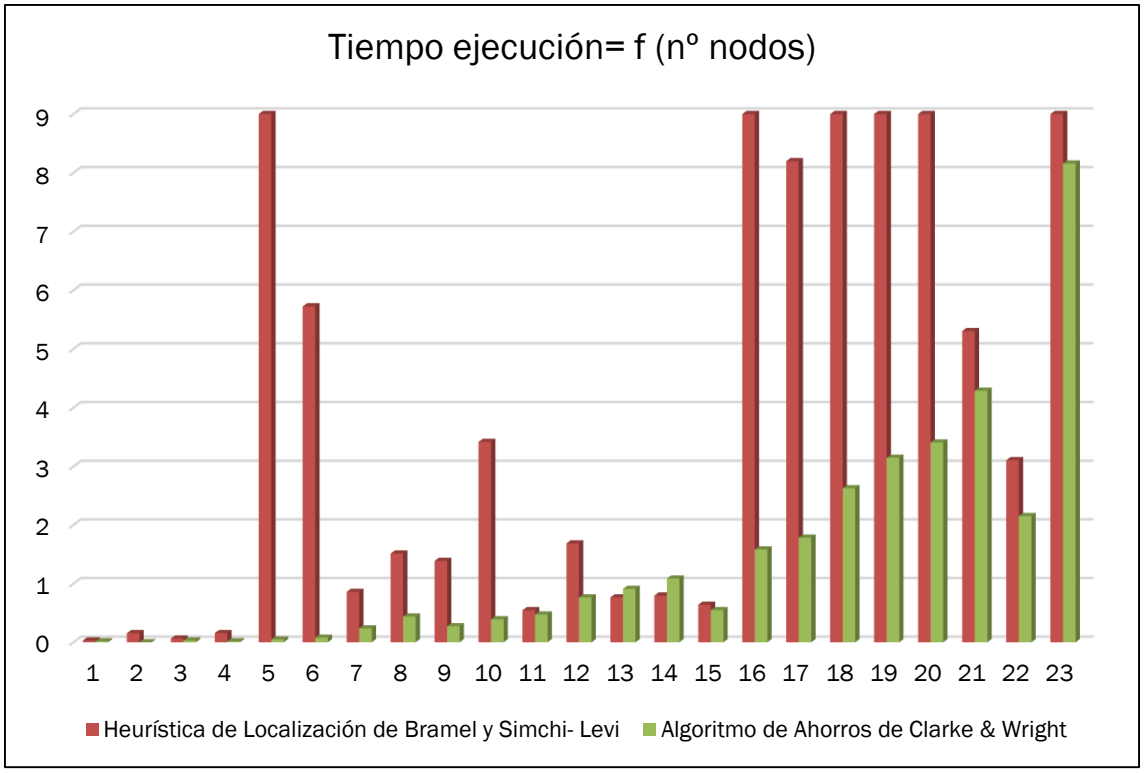

Gráfico 15. Tiempo de ejecución como función del número de nodos. Comparación Heurística de Localización de Bramel y Simchi- Levi con Algoritmo de Ahorros original. (Elaboración propia).

6.3. Métodos de mejora

6.3.1. Mejora exacta mediante restricciones de tipo Tucker- Miller- Zemlin

6.3.1.1. Descripción

En este apartado se va a aplicar un método de mejora para el Algoritmo de Ahorros de Tucker- Miller- Zemlin original. Dicho proceso consiste en intentar mejorar cada una de las rutas que salieron con dicho algoritmo mediante la formulación del problema a través del modelo de dos índices basado en Tucker- Miller- Zemlin que se vio en el Capítulo 2. Como el número de nodos de cada ruta es pequeño, es de esperar que dicho modelo tarde poco en ejecutarse.

6.3.1.2. Experiencia computacional

Los resultados que se obtienen al aplicar la mejora exacta mediante restricciones de Tucker. Miller- Zemlin al Algoritmo de Ahorros son los que figuran en la Tabla 16.

Datos	Mejora exacta del Algoritmo de Ahorros mediante formulación exacta de Tucker- Miller-	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	119	0,016
eilon_13_4	290	0,016
eilon_22_4	387	0,015
eilon_23_3	569	0,031
eilon_30_3	507	10,531
eilon_31_7	1263	0,093
A-n32-k5	830	0,749
eilon_33_4	843	0,499
A-n34-k5	793	0,112
A-n37-k6	969	0,946
A-n39-k6	845	0,168
A-n44-k7	1001	0,133
A-n46-k7	939	0,361
A-n48-k7	1098	0,269
eilon_51_5	575	0,171
A-n53-k7	1125	0,276
A-n55-k9	1109	0,178
A-n60-k9	1364	0,339
A-n63-k10	1339	0,421
A-n65-k9	1251	0,265
A-n69-k9	1187	0,468
eilon_76_7	726	0,375
A-n80-k10	1810	1,17

Tabla 16. Resultados implementación. Mejora del Algoritmo de Ahorros mediante formulación exacta de Tucker- Miller- Zemlin. (Elaboración propia).

Se vio en al Apartado 6.1.6.1 que el Algoritmo de Ahorros proporciona mejores resultados que los métodos exactos. Al ser este método que vamos a aplicar una mejora de éste, es evidente que también proporciona mejores soluciones que los modelos que se plantearon.

Además, al ser un método de mejora, está claro que también proporciona mejores resultados que el Algoritmo de Ahorros sin la mejora implementada.

Por ello, comparamos el Algoritmo de Ahorros de Clarke & Wright mejorado mediante Tucker- Miller- Zemlin con la Heurística de Localización de Bramel y Simchi Levi. El primero, proporciona mejores resultados en el 42 % de los casos (Gráfico 16).

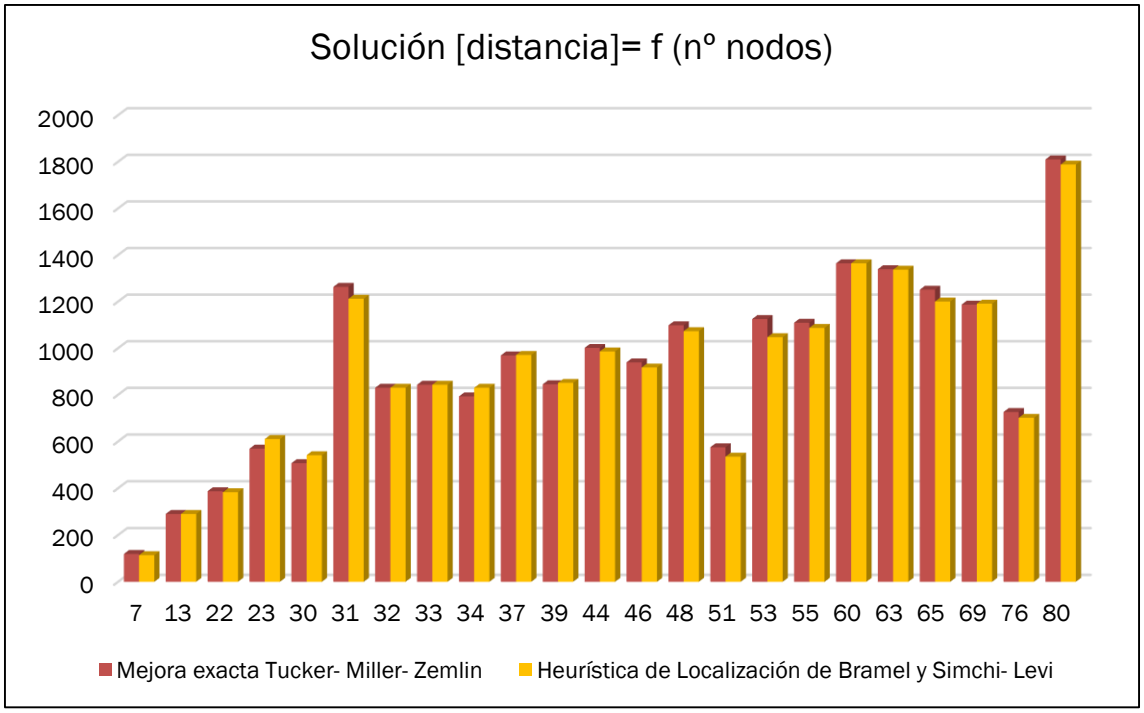

Gráfico 16. Distancia recorrida como función del número de nodos. Comparación mejora exacta del Algoritmo de Ahorros con Heurística de Localización de Bramel y Simchi- Levi. (Elaboración propia).

Si se comparan los tiempos de ejecución, se puede apreciar como utilizan en la mayoría de las ocasiones tiempos similares (Gráfico 17). Es de resaltar, que para los casos de 60, 63 y 80 nodos, la heurística del concentrador emplea bastante más tiempo. Este hecho podemos no tenerlo en cuenta porque aunque tardó ese tiempo en dar una solución de 1364, 1337 y 1788 respectivamente, puede que a esa solución llegase en un tiempo menor, esto es, que encontrase dicha solución en menor tiempo aunque siguiese buscando soluciones (sin encontrar una mejor) hasta los 200, 84 y 245 segundos.

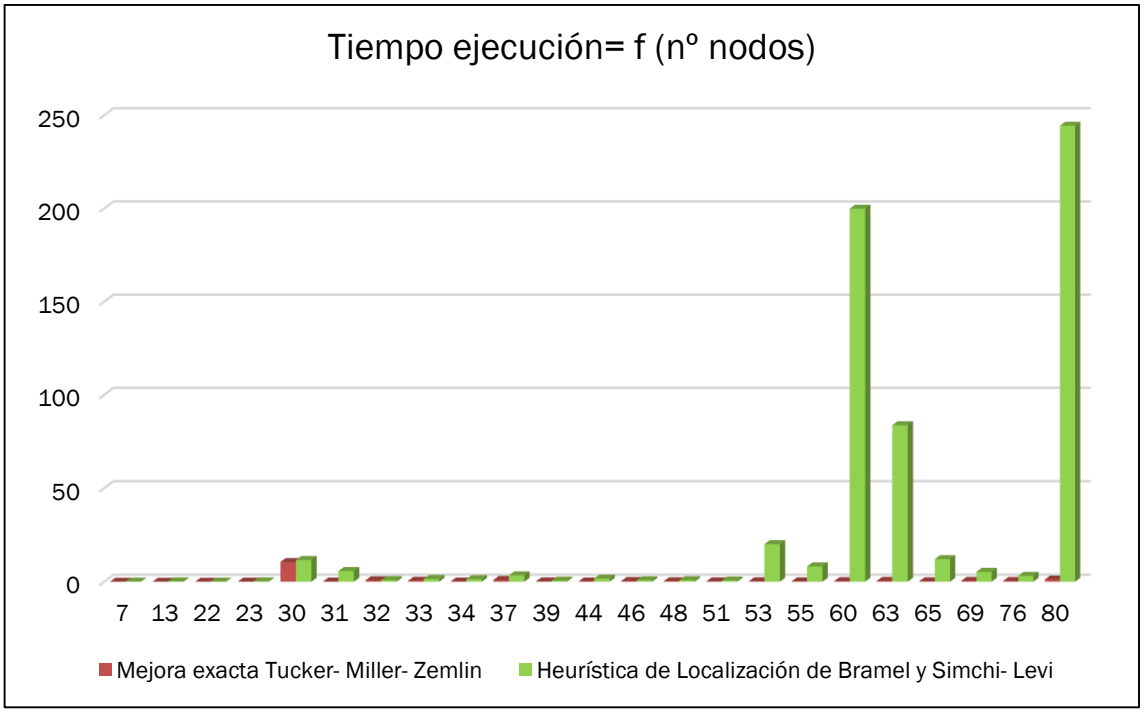

Gráfico 17. Tiempo de ejecución como función del número de nodos. Comparación mejora exacta del Algoritmo de Ahorros con Heurística de Localización de Bramel y Simchi- Levi. (Elaboración propia).

6.3.2. Mejora 2-opt

6.3.2.1. Descripción

La heurística 2- opt está basada en el hecho de que para un problema euclídeo, si dos aristas se cruzan, pueden ser sustituidas por otras dos que no se crucen, mejorando el recorrido. Además, sólo existe una posible conexión (Figura 21).

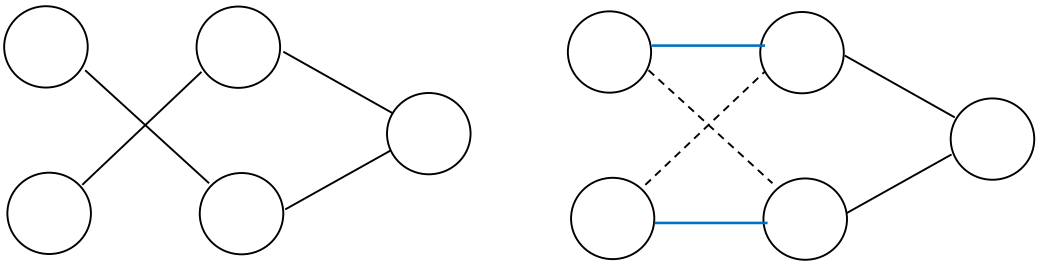

Figura 21. Único 2- intercambio posible para los arcos marcados. (Elaboración propia).

Este algoritmo parte de una solución obtenida generalmente mediante un método constructivo y funciona de la siguiente manera: sean dos arcos de una misma ruta, cuyos puntos son (i, s_i) y (j, s_j) tal que $i \neq j$ y $j \neq s_i$ se sustituyen los tramos del recorrido $i \rightarrow s_i$ y $j \rightarrow s_j$ por otros dos arcos, que tienen que ser necesariamente $i \rightarrow$ j y $s_i \rightarrow s_j$. Para que este cambio pueda ser factible, es necesario cambiar el arco (j, s_i) de sentido. Gráficamente se muestra en las Figura 22 y Figura 23.

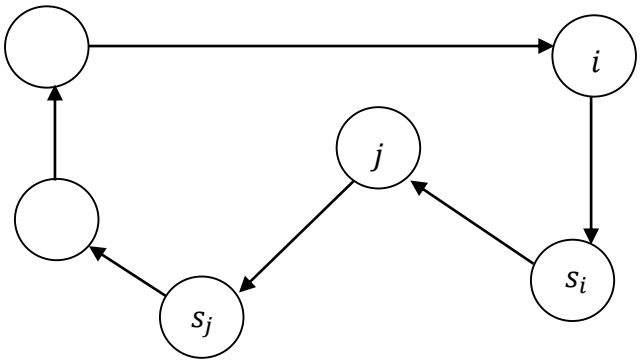

Figura 22. Estado inicial heurística 2- opt. (Elaboración propia).

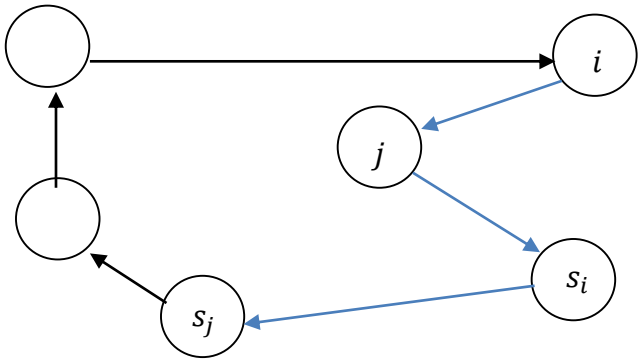

Figura 23. Intercambio 2- opt. (Elaboración propia).

La mejora obtenida con este método es:

$$mejora = d(i, s_i) + d(j, s_j) - d(i, j) - d(s_i, s_j) \quad (43)$$

El método se termina cuando no se mejora o cuando la mejora es menor que una tolerancia establecida por el usuario. En ese momento, la solución que hemos obtenido es un óptimo local, siendo toda solución del entorno peor que ésta.

6.3.2.2. Pasos a seguir

Paso 1: escoger una ruta de las obtenidas por un método constructivo.

Paso 2: para cada par de nodos tal que $i \neq j$ y $j \neq s_i$, calcular la mejora mediante la formula.

Paso 3: si la mejora es mayor que la mejora máxima obtenida hasta el momento, realizar el intercambio 2- opt. Si por el contrario, no es mayor o es menor que la cota que se ha establecido, el método finaliza.

6.3.2.3. Consideraciones

1.- Señalar que, la solución que se obtiene con esta heurística depende del nodo de inicio que se haya elegido. (Figura 24).

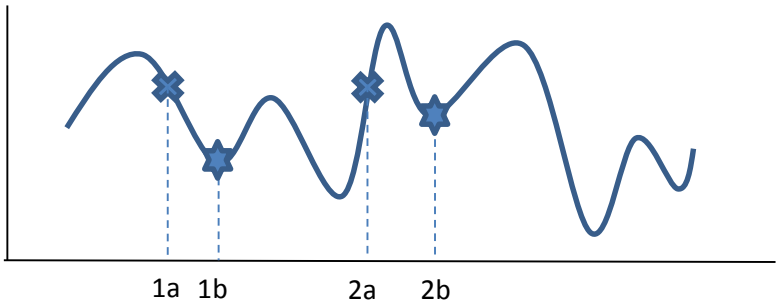

Figura 24. Dependencia de solución heurística 2- opt con la solución inicial. (Elaboración propia).

Es decir, si la solución de la que partimos es la 1a, mediante la 2- opt conseguiremos el óptimo 1b; mientras que si la solución inicial es la 2a, el óptimo que obtenemos es el 2b.

2.- Además, según está explicada, la heurística 2- opt es solo válida para el caso simétrico (PRVCS), pues se está suponiendo que la distancia del tramo donde se cambia la dirección no se modifica al cambiar ésta.

3.- No se va a implementar la heurística 2- opt porque nunca se van a obtener mejores resultados que con la mejora exacta (Apartado 6.3.1).

6.4. Conclusiones

Se ha demostrado que:

1. Comparado con los métodos exactos, los heurísticos proporcionan soluciones bastante buenas en tiempo de ejecución bastante menores.
2. No merecen la pena las modificaciones del Algoritmo de Ahorros original, pues aunque con la introducción de uno y dos parámetros en la fórmula de ahorros sí que se consiguen mejores resultados, el tiempo computacional que emplean no lo compensa.
3. La heurística de Localización de Bramel y Simchi- Levi proporciona mejores soluciones que el Algoritmo de Ahorros y que la mejora de éste mediante formulación exacta de Tucker- Miller- Zemlin.

Por todo ello, podemos concluir que, el método de resolución con el que se consiguen mejores resultados es mediante la Heurística de Localización de Bramel y Simchi- Levi.

CAPÍTULO 7:

Metaheurística GRASP para el CVRP

Una de las metaheurísticas más exitosas que han aparecido en los últimos años es el GRASP, un método multi- arranque diseñado para resolver problemas difíciles de optimización combinatoria. Este procedimiento de búsqueda miope aleatorizado y adaptativo (GRASP por sus siglas en inglés) permite encontrar soluciones de buena calidad pero no necesariamente óptimas. Se puede consultar *"Handbook of metaheuristics"* (Fred Glover, Gary A. Kochenberger).

7.1. Descripción

La metaheurística GRASP se trata de un método iterativo, en el que cada iteración consiste en dos fases: una fase constructiva y una búsqueda local.

Primera fase- Fase constructiva:

Durante esta fase se introduce la aleatoriedad. Existen varias formas de introducirla, pero la más común es el uso de una lista restringida de candidatos (RCL). Dicha lista, de tamaño K, contiene el conjunto de elementos candidatos a añadir a la solución y, de entre dichos candidatos, el elemento a añadir se elige de forma aleatoria. Este proceso se repite hasta tener una solución final.

Trasladado al CVRP, se trata de, en vez de escoger en cada iteración el máximo ahorro, agrupar los ahorros (ya ordenados de mayor a menor) en listas de tamaño K, y escoger un elemento de la lista de manera aleatoria. Posteriormente se volverían a agrupar los ahorros en listas de ese tamaño y se volvería a escoger uno de manera aleatoria, y así sucesivamente.

Segunda fase- Fase de mejora:

Una vez se tiene la solución final que se acaba de comentar, se hace la mejora. Ésta puede llevarse a cabo mediante cualquiera de los métodos de mejora existentes, ya sean exactos o heurísticos: 2- opt, 3- opt, redes, Tucker- Miller- Zemlin, etc. Para la implementación realizada se ha usado este último.

El proceso anterior (fase constructiva más fase de mejora), se repine N veces, obteniendo N soluciones, de entre las cuales se cogerá la mejor.

7.2. Principal ventaja

Tal y como se puede apreciar, dicho método de resolución, cumple los principios de diversificación e intensificación, que tan importantes son en optimización. El primero, debido a que el punto de inicio es aleatorio y por tanto, no siempre se comienza en el mismo punto inicial como ocurría en la heurística de Clarke & Wright que siempre se partía del mayor ahorro; y el segundo, mediante la mejora.

7.3. Experiencia computacional

A la hora de implementar la metaheurística GRASP:

- Se ha implementado sin la fase de mejora y con ella para el Algoritmo de Ahorros de Clarke & Wright.
- Se han incluído como parámetros el número de iteraciones y el tamaño de la lista restringida de candidatos.

7.3.1. Comparación GRASP en función del tamaño de la lista restringida de candidatos

Si se ejecuta para $N = 40$ y $K \in \{4,8,16,32\}$, se obtienen los resultados que aparecen en la Tabla 17, Tabla 18, Tabla 19 y Tabla 20.

Datos	GRASP sin mejora		GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,016	114	0,016
eilon_13_4	300	0,031	290	0,031
eilon_22_4	387	0,156	375	0,156
eilon_23_3	595	0,156	577	0,156
eilon_30_3	533	0,406	514	0,406
eilon_31_7	1263	0,858	1263	0,843
eilon_33_4	863	0,905	842	0,905
eilon_51_5	570	3,137	556	3,137
eilon_76_7	752	14,151	718	14,151

Tabla 17. Resultados implementación. GRASP: N=40, K=4. (Elaboración propia).

Datos	GRASP sin mejora		GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	120	0,016	114	0,016
eilon_13_4	308	0,031	290	0,038
eilon_22_4	429	0,156	387	0,156
eilon_23_3	613	0,14	575	0,14
eilon_30_3	522	0,406	504	0,406
eilon_31_7	1267	0,842	1248	0,842
eilon_33_4	885	0,889	841	0,889
eilon_51_5	582	3,12	548	3,104
eilon_76_7	747	14,339	706	14,323

Tabla 18. Resultados implementación. GRASP: N=40, K=8. (Elaboración propia).

Datos	GRASP sin mejora		GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	119	0,015	116	0,015
eilon_13_4	322	0,032	292	0,032
eilon_22_4	427	0,141	394	0,141
eilon_23_3	575	0,141	571	0,141
eilon_30_3	539	0,39	514	0,39
eilon_31_7	1281	0,943	1252	0,843
eilon_33_4	885	0,889	841	0,889
eilon_51_5	608	3,152	548	3,152
eilon_76_7	722	14,712	7622	14,712

Tabla 19. Resultados implementación. GRASP: N=40, K=16. (Elaboración propia).

Datos	GRASP sin mejora		GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	210	0,015	172	0,016
eilon_13_4	316	0,031	302	0,031
eilon_22_4	488	0,141	416	0,141
eilon_23_3	633	0,14	599	0,14
eilon_30_3	577	0,39	526	0,39
eilon_31_7	1302	0,811	1261	0,811
eilon_33_4	1013	0,976	868	0,874
eilon_51_5	597	3,151	568	3,151
eilon_76_7	739	14,386	735	14,386

Tabla 20. Resultados implementación. GRASP: N=40, K=32. (Elaboración propia).

Como se puede apreciar en las tablas anteriores, la diferencia de tiempos entre el GRASP sin la fase de mejora (o lo que sería lo mismo, la heurística de Clarke & Wright aleatorizada) y el GRASP con la fase de mejora mediante formulación exacta de Tucker- Miller- Zemlin es insignificante en todos los casos, por lo que está claro que merece la pena incluir dicha fase de mejora.

Si representamos las soluciones (Gráfico 18a, 18b), se puede apreciar como a medida que el tamaño de la RCL aumenta, las soluciones del método GRASP son peores. Por tanto, se puede deducir que éste proporciona mejores soluciones cuanto más pequeña sea la RCL.

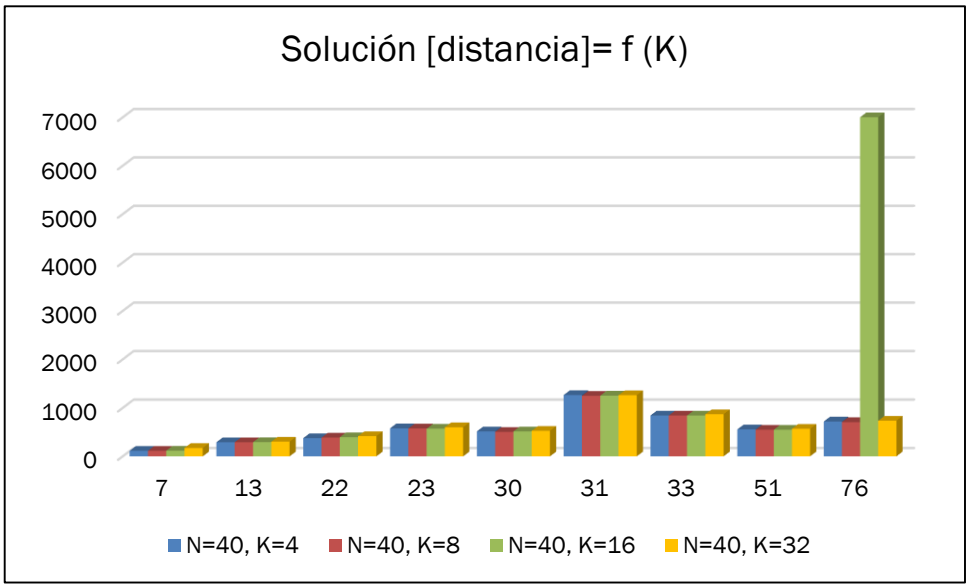

Gráfico 18a. Distancia recorrida como función del número de nodos. GRASP: comparación tamaño K. (Elaboración propia).

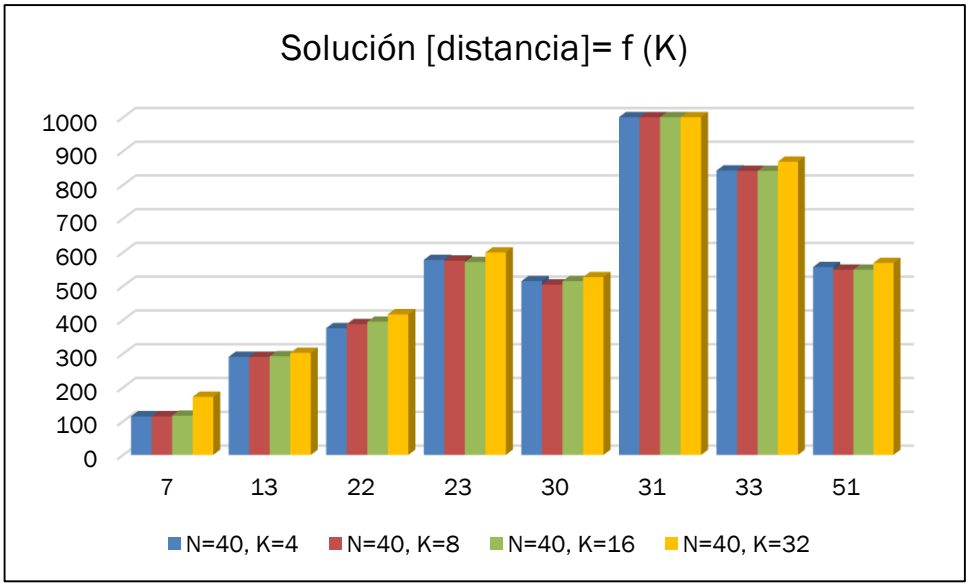

Gráfico 18b. Ampliación Gráfico 18a. (Elaboración propia).

7.3.2. Comparación GRASP en función del número de iteraciones

Debido a los razonamientos anteriores, con el GRASP con la fase de mejora y con un tamaño de la lista restringida de candidatos igual a cuatro, y lo implementamos para valores de $N \in \{10,20,40,80,160\}$, obteniéndose los valores de la Tabla 21, Tabla 22, Tabla 23, Tabla 24 y Tabla 25.

Datos	GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,016
eilon_13_4	290	0,016
eilon_22_4	377	0,046
eilon_23_3	574	0,047
eilon_30_3	511	0,124
eilon_31_7	1263	0,25
eilon_33_4	841	0,256
eilon_51_5	569	1,014
eilon_76_7	726	4,728

Tabla 21. Resultados implementación. GRASP: N=10, K=4. (Elaboración propia).

Datos	GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,015
eilon_13_4	290	0,015
eilon_22_4	375	0,078
eilon_23_3	579	0,093
eilon_30_3	506	0,234
eilon_31_7	1263	0,437
eilon_33_4	841	0,468
eilon_51_5	562	1,747
eilon_76_7	724	8,004

Tabla 22. Resultados implementación. GRASP: N=20, K=4. (Elaboración propia).

Datos	GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,016
eilon_13_4	290	0,031
eilon_22_4	375	0,156
eilon_23_3	577	0,156
eilon_30_3	514	0,406
eilon_31_7	1263	0,843
eilon_33_4	842	0,905
eilon_51_5	556	3,137
eilon_76_7	718	14,151

Tabla 23. Resultados implementación. GRASP: N=40, K=4. (Elaboración propia).

Datos	GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,016
eilon_13_4	290	0,063
eilon_22_4	375	0,296
eilon_23_3	569	0,313
eilon_30_3	512	0,78
eilon_31_7	1249	1,654
eilon_33_4	841	1,685
eilon_51_5	558	5,846
eilon_76_7	713	27,96

Tabla 24. Resultados implementación. GRASP: N=80, K=4. (Elaboración propia).

Datos	GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,031
eilon_13_4	290	0,025
eilon_22_4	375	0,546
eilon_23_3	572	0,53
eilon_30_3	510	1,513
eilon_31_7	1247	3,245
eilon_33_4	841	3,322
eilon_51_5	551	11,513
eilon_76_7	714	52,363

Tabla 25. Resultados implementación. GRASP: N=160, K=4. (Elaboración propia).

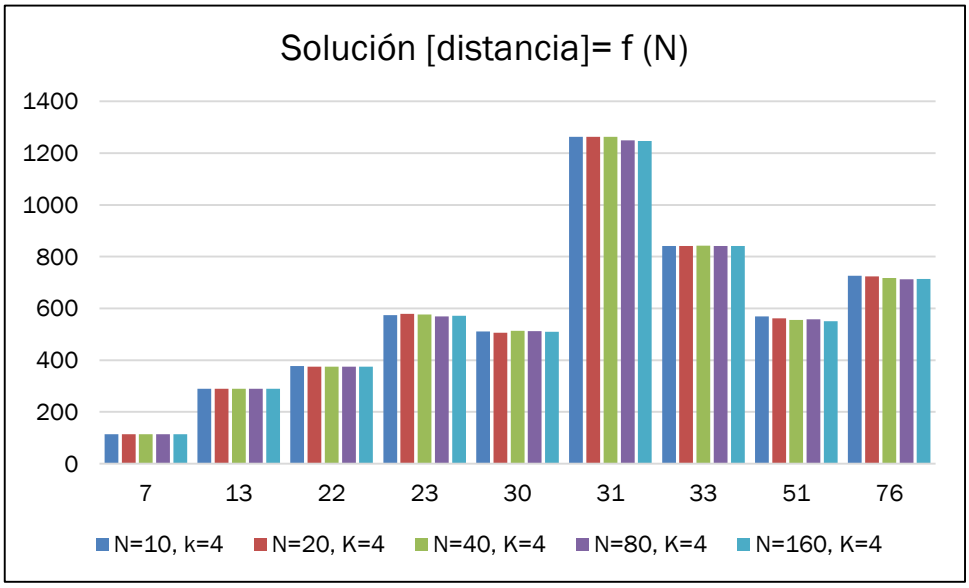

Gráfico 19. Distancia recorrida como función del número de nodos. GRASP: comparación tamaño N (Elaboración propia).

Como los tiempos son muy similares (la máxima diferencia ocurre para el caso de 76 nodos, con 48 segundos), para comparar el número de iteraciones nos fijaremos en la distancia del recorrido. En el Gráfico 19 se puede apreciar como no hay casi diferencia en la solución final al aumentar el tamaño de N.

7.3.3. Comparación GRASP con Heurística de Localización de Bramel y Simchi- Levi

Debido a los razonamientos de los dos apartados anteriores, para el siguiente apartado, se ha decidido escoger un valor de $N = 10$ y $K = 4$, obteniendo los valores que aparecen en la Tabla 26.

Datos	GRASP con mejora	
	Mejor solución factible (km)	Tiempo ejecución (s)
eilon_7_2	114	0,016
eilon_13_4	290	0,016
eilon_22_4	377	0,046
eilon_23_3	574	0,047
eilon_30_3	511	0,124
eilon_31_7	1263	0,25
A-n32-k5	809	0,203
eilon_33_4	841	0,256
A-n34-k5	799	0,281
A-n37-k6	977	0,453
A-n39-k6	848	0,468
A-n44-k7	958	0,796
A-n46-k7	939	0,92
A-n48-k7	1104	1,201
eilon_51_5	569	1,014
A-n53-k7	1056	1,623
A-n55-k9	1101	1,732
A-n60-k9	1373	2,98
A-n63-k10	1349	3,496
A-n65-k9	1232	3,199
A-n69-k9	1195	3,839
eilon_76_7	726	4,728
A-n80-k10	1811	8,954

Tabla 26. Resultados completos implementación. GRASP: N=10, K=4. (Elaboración propia).

Señalar que, se ha decido comparar con dicha heurística porque se demostró en el Capítulo 6 que era la que mejores resultados proporcionaba. De esta forma, el GRASP proporciona mejores resultados que la heurística en el 50 % de los casos muestreados (Gráfico 20), pero emplea tiempos computacionales algo inferiores (Gráfico 21), por lo que se puede concluir que la metaheurística GRASP es mejor.

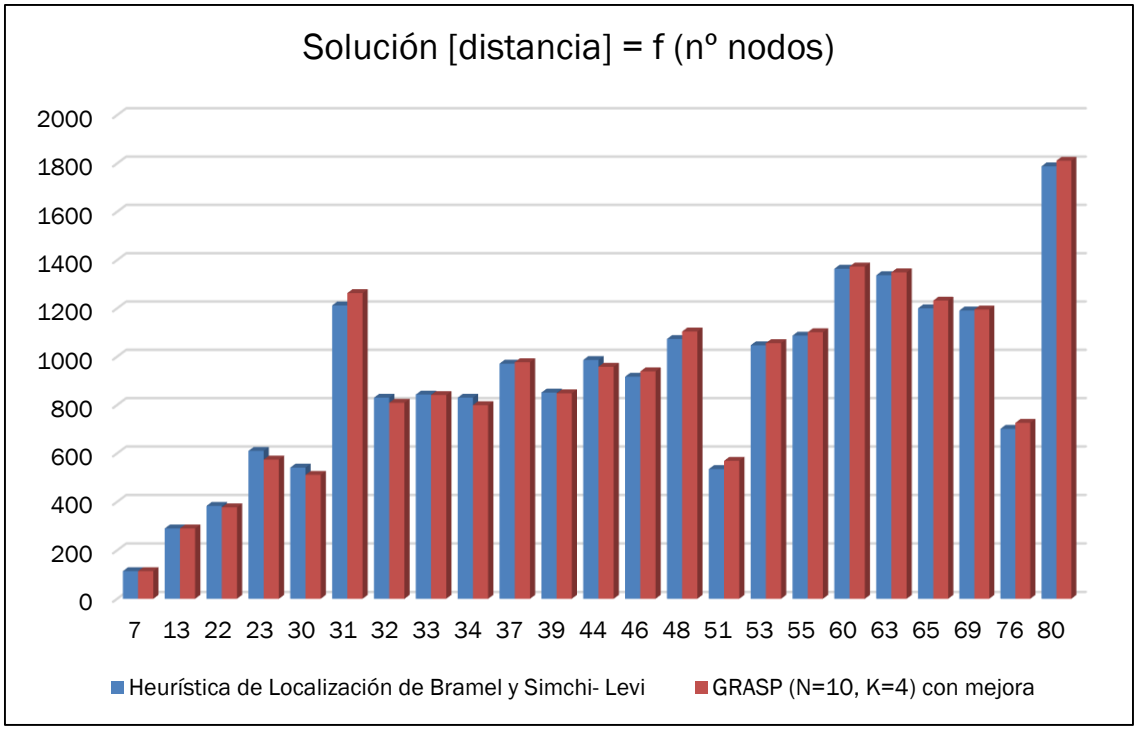

Gráfico 20. Distancia recorrida como función del número de nodos. Comparación GRASP con Heurística de Localización de Bramel y Simchi- Levi (Elaboración propia).

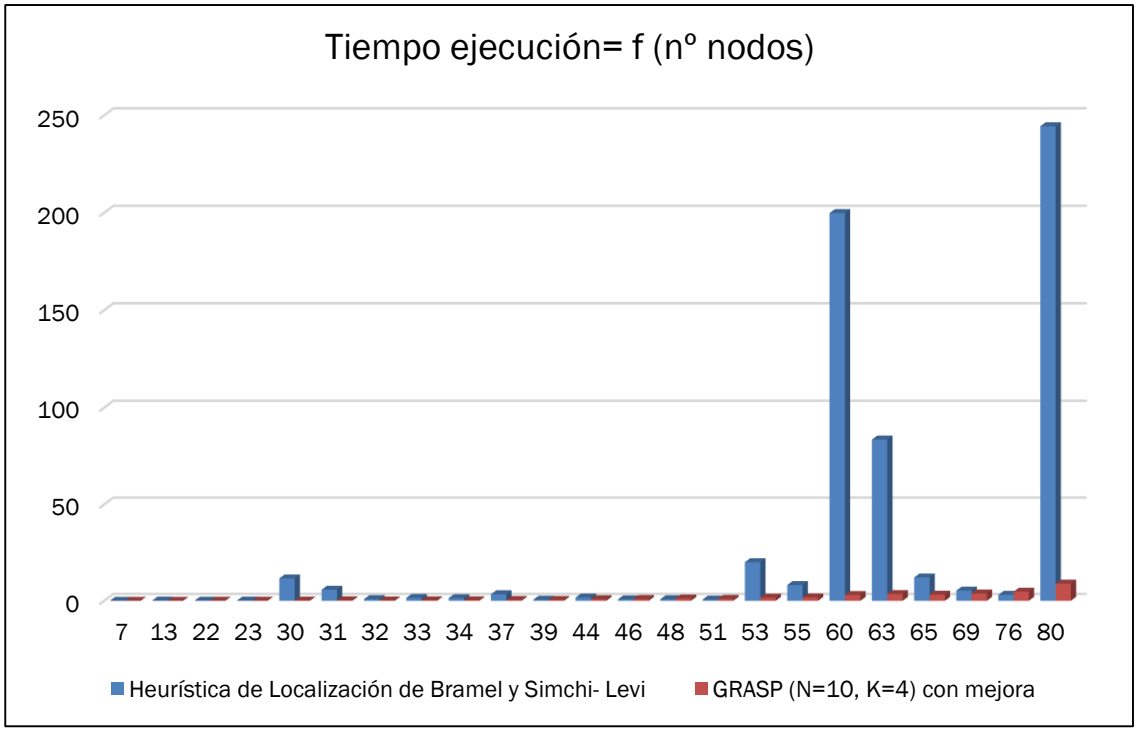

Gráfico 21. Tiempo de ejecución como función del número de nodos. Comparación GRASP con Heurística de Localización de Bramel y Simchi- Levi (Elaboración propia).

CAPÍTULO 8:

Conclusiones finales y futuras líneas de trabajo

8.1. Conclusiones finales

El Problema de Rutas de Vehículos es uno de los problemas más importantes de optimización debido a la gran cantidad de casos en los que es aplicable así como la ventaja competitiva que se conseguiría si se gestionasen las rutas de forma óptima.

A partir del problema básico (Problema de Rutas de Vehículos Capacitado) cuya complejidad, como se ha demostrado, es enorme, existen numerosas variantes (con ventanas de tiempo, con inventarios, con retornos, etc), que dificultarían enormemente su estudio.

Se pueden emplear varios métodos para su resolución. Los exactos, que proporcionan soluciones muy buenas pero tiempos elevados, y los aproximados que dan buenas soluciones (aunque no óptimas) en tiempos menores.

Además, los dos mejores métodos aproximados han resultado ser la Heurística de Localización de Bramel y Simchi- Levi y la metaheurística GRASP, aunque el Algoritmo de Ahorros proporciona soluciones bastante buenas que mejoran al añadir la formulación exacta de Tucker- Miller- Zemlin.

La resolución de los problemas de rutas que hoy en día tienen muchas empresas mediante cualquiera de los métodos planteados permitiría, sin duda, que sus constes disminuyeran considerablemente.

8.2. Futuras líneas de trabajo

Como se ha visto a lo largo del presente documento, desde que por primera vez se oyó hablar del Problema de Rutas de Vehículos, han surgido numerosas variantes y métodos de resolución. Debido a la complejidad y grandeza de dicho problema, este Trabajo Fin de Grado se ha centrado en el Problema de Rutas de Vehículos Capacitado, que puede ser el punto de partida para posibles estudios futuros. Algunos de ellos podrían ser:

- Problemas de Rutas de Vehículos con ventanas de tiempo.
- Problemas de Rutas de Vehículos periódico.

- Problemas de Rutas de Vehículos con inventarios.
- Problemas de Rutas de Vehículos con retornos.
- Aplicación al transporte escolar.
- Métodos de resolución exactos y aproximados para el TSP.
- Variantes del TSP.
- Resolución del CVRP mediante las heurísticas y metaheurísticas que existen y no se han desarrollado en el presente documento.

ANEXOS

Anexo 1: Programación de modelos exactos para flota homogénea

Modelo basado en redes

```
model "Modelo basado en redes”
uses "mmxprs";
uses "mmsystem";
parameters
        archivo_datos="C:/eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos=1..n
        x:array(nodos,nodos) of mpvar
        y:array(nodos,nodos) of mpvar
        u:array(nodos) of mpvar
        dem:array(nodos)of integer
        dist:array(nodos,nodos) of integer
        cap:integer
        tiempo_calculo = -200
        i1, final, iter,ruta:integer
        inicial = 1
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================MODELO================
objetivo:=sum(i,j in nodos|i<>j)dist(i,j)*x(i,j)
forall(i,j in nodos)x(i,j)is_binary
forall(i in nodos|i<>1)res_1(i):=sum(j in nodos|i<>j)x(i,j)=1
forall(j in nodos|j<>1)res_2(j):=sum(i in nodos|i<>j)x(i,j)=1
res_3:=sum(j in nodos|j<>1)x(1,j)=m
res_4:=sum(j in nodos|j<>1)x(j,1)=m
forall(i in nodos|i<>1)res_5(i):=sum(j in nodos|i<>j)y(i,j)-sum(j in nodos|i<>j)y(j,i)=dem(i)
forall(i,j in nodos|i<>j)res_6(i,j):=y(i,j)<=cap*x(i,j)
!===============OPTIMIZACIÓN===============
starttime:=gettime
setparam("XPRS_MAXTIME",tiempo_calculo)
minimize(objetivo)
nvar:= getparam("XPRS_cols")
nres:= getparam("XPRS_rows")
```

```
writeln("Numero de variables: ",nvar,", numero de restricciones: ",nres)
mip_status:= getprobstat
mejor_cota:= getparam("XPRS_bestbound")
nodos_BB:=getparam("XPRS_nodes")
if(getobjval<>0)then
    hueco_final:= 100*(getobjval-mejor_cota)/getobjval
end-if
!================RESULTADOS===============
writeln("\nSolución:")
if(mip_status=XPRS_INF)then
        writeln("Problema no factible")
        exit(0)
end-if
if(mip_status=XPRS_OPT)then
        writeln("Solución óptima")
elif(mip_status=XPRS_UNF)then
        writeln("Solución entera factible")
end-if
writeln("Valor objetivo = distancia = ",getobjval)
writeln("Mejor cota= ",mejor_cota)
writeln("Hueco final= ",strfmt(hueco_final,6,2),"%")
writeln("Nodos examinados = ",nodos_BB)
writeln("Tiempo en segundos = ",gettime-starttime,"\n\n")
writeln("Matriz de soluciones:")
forall(i in nodos) do
writeln
        forall(j in nodos)write("\t",x(i,j).sol)
end-do
writeln("\n\nLas variables distintas de cero son :")
forall (i,j in nodos|x(i,j).sol>=0.99) do
        writeln("x(",i,",",j,")")
end-do
ruta:=0
forall(i in nodos|i<>inicial)do
        if(x(inicial,i).sol>=0.99)then
                ruta:=ruta+1
                carga:=dem(i)
                iter:=1
                write("\n  ruta ",ruta,": ",inicial,"-",i)
                i1:=i
                final:=0
                while(final=0)do
                        if(i1=inicial)then
                                final:=1
                        else
                                forall(j in nodos)do
                                        if(x(i1,j).sol>=0.999)then
                                                carga:=carga+dem(j)
                                                i1:=j
                                                write("-",i1)
                                                if(i1=inicial)then break;end-if
                                        end-if
                                end-do
                        end-if
                end-do
        write(", carga = ",carga)
end-do
end-model
```

Modelo de dos índices basado en Tucker- Miller- Zemlin

```
model "Modelo de dos índices basado en Tucker- Miller- Zemlin”
uses "mmxprs";
uses "mmsystem";
parameters
        archivo_datos="C:/eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos=1..n
        x:array(nodos,nodos) of mpvar
        y:array(nodos,nodos) of mpvar
        u:array(nodos) of mpvar
        dem:array(nodos)of integer
        dist:array(nodos,nodos) of integer
        cap:integer
        tiempo_calculo = -200
        i1, final, iter,ruta:integer
        inicial = 1
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================MODELO================
objetivo:=sum(i,j in nodos|i<>j)dist(i,j)*x(i,j)
forall(i,j in nodos)x(i,j)is_binary
forall(i in nodos|i<>1)res_1(i):=sum(j in nodos|i<>j)x(i,j)=1
forall(j in nodos|j<>1)res_2(j):=sum(i in nodos|i<>j)x(i,j)=1
res_3:=sum(j in nodos|j<>1)x(1,j)=m
res_4:=sum(j in nodos|j<>1)x(j,1)=m
forall(i,j in nodos|i<>j and i<>1 and j<>1 and dem(i)+dem(j)<=cap)res_5(i,j):=u(i)-
u(j)+cap*x(i,j)<=cap-dem(j)
forall(i in nodos|i<>1)do
        res_6(i):=dem(i)<=u(i)
        res_7(i):=u(i)<=cap
end-do
!===============OPTIMIZACIÓN===============
starttime:=gettime
setparam("XPRS_MAXTIME",tiempo_calculo)
minimize(objetivo)
nvar:= getparam("XPRS_cols")
nres:= getparam("XPRS_rows")
writeln("Numero de variables: ",nvar,", numero de restricciones: ",nres)
mip_status:= getprobstat
mejor_cota:= getparam("XPRS_bestbound")
nodos_BB:=getparam("XPRS_nodes")
```

```
if(getobjval<>0)then
    hueco_final:= 100*(getobjval-mejor_cota)/getobjval
end-if
!================RESULTADOS================
writeln("\nSolución:")
if(mip_status=XPRS_INF)then
        writeln("Problema no factible")
        exit(0)
end-if
if(mip_status=XPRS_OPT)then
        writeln("Solución óptima")
elif(mip_status=XPRS_UNF)then
        writeln("Solución entera factible")
end-if
writeln("Valor objetivo = distancia = ",getobjval)
writeln("Mejor cota= ",mejor_cota)
writeln("Hueco final= ",strfmt(hueco_final,6,2),"%")
writeln("Nodos examinados = ",nodos_BB)
writeln("Tiempo en segundos = ",gettime-starttime,"\n\n")
writeln("Matriz de soluciones:")
forall(i in nodos) do
        writeln
        forall(j in nodos)write("\t",x(i,j).sol)
end-do
writeln("\n\nLas variables distintas de cero son :")
forall (i,j in nodos|x(i,j).sol>=0.99) do
                writeln("x(",i,",",j,")")
end-do
ruta:=0
forall(i in nodos|i<>inicial)do
        if(x(inicial,i).sol>=0.99)then
                ruta:=ruta+1
                carga:=dem(i)
                iter:=1
                write("\n  ruta ",ruta,": ",inicial,"-",i)
                i1:=i
                final:=0
                while(final=0)do
                        if(i1=inicial)then
                                final:=1
                        else
                                forall(j in nodos)do
                                        if(x(i1,j).sol>=0.999)then
                                                carga:=carga+dem(j)
                                                i1:=j
                                                write("-",i1)
                                                if(i1=inicial)then break;end-if
                                        end-if
                                end-do
                        end-if
                end-do
        write(", carga = ",carga)
end-do
end-model
```

Modelo de tres índices basado en Tucker- Miller- Zemlin

```
model "Modelo de tres índices basado en Tucker- Miller- Zemlin”
uses "mmxprs";
uses "mmsystem";
parameters
        archivo_datos="C:/eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos=1..n
        vehiculos=1..m
        x:array(nodos,nodos,vehiculos) of mpvar
        y:array(nodos,nodos,vehiculos) of mpvar
        u:array(nodos,vehiculos) of mpvar
        dem:array(nodos)of integer
        dist:array(nodos,nodos) of integer
        cap:integer
        tiempo_calculo = -200
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
!=================MODELO================
objetivo:=sum (i in nodos, j in nodos, h in vehiculos)dist(i,j)*x(i,j,h)
forall(i in nodos|i>1)res_1(i):=sum(h in vehiculos)y(i,h)=1
res_2:=sum(h in vehiculos)y(1,h)<=m
forall(i in nodos, h in vehiculos|i>1)res_3(i,h):=dem(i)<=u(i,h)
forall(i in nodos, h in vehiculos|i>1)res_4(i,h):=u(i,h)<=cap
forall(i in nodos, j in nodos, h in vehiculos|i>1 and j>1 and i<>j and
dem(i)+dem(j)<=cap)res_5(i,j,h):= u(i,h)-u(j,h)+cap*x(i,j,h)<=cap-dem(j)
forall(i in nodos,h in vehiculos)res_5(i,h):=sum(j in nodos)x(i,j,h)=y(i,h)
forall(i in nodos,h in vehiculos)res_6(i,h):=sum(j in nodos)x(j,i,h)=y(i,h)
forall(h in vehiculos)res_7(h):=sum(i in nodos)dem(i)*y(i,h)<=cap
forall(i in nodos, j in nodos,h in vehiculos)x(i,j,h) is_binary
forall(i in nodos, h in vehiculos)y(i,h)is_binary
!===============OPTIMIZACIÓN===============
starttime:=gettime
setparam("XPRS_MAXTIME",tiempo_calculo)
minimize(objetivo)
nvar:= getparam("XPRS_cols")
nres:= getparam("XPRS_rows")
writeln("Numero de variables: ",nvar,", numero de restricciones: ",nres)
mip_status:= getprobstat
mejor_cota:= getparam("XPRS_bestbound")
nodos_BB:=getparam("XPRS_nodes")
if(getobjval<>0)then
    hueco_final:= 100*(getobjval-mejor_cota)/getobjval
```

```
end-if
!================RESULTADOS================
writeln("\nSolución:")
if(mip_status=XPRS_INF)then
        writeln("Problema no factible")
        exit(0)
end-if
if(mip_status=XPRS_OPT)then
        writeln("Solución óptima")
elif(mip_status=XPRS_UNF)then
        writeln("Solución entera factible")
end-if
writeln("Valor objetivo = distancia = ",getobjval)
writeln("Mejor cota= ",mejor_cota)
writeln("Hueco final= ",strfmt(hueco_final,6,2),"%")
writeln("Nodos examinados = ",nodos_BB)
writeln("Tiempo en segundos = ",gettime-starttime,"\n\n")
writeln("\n\nLas variables distintas de cero son :")
forall (i in nodos,j in nodos,h in vehiculos|x(i,j,h).sol>=0.99) do
        writeln("z(",i,",",j,",",h,")")
end-do
writeln("\nDistancia total recorrida = ",getobjval)
writeln("\nAsignación de nodos a vehículos:")
forall(i in nodos,h in vehiculos|y(i,h).sol>=0.99)writeln("t(",i,",",h,") = ",y(i,h).sol)
forall(h in vehiculos)do
        writeln("\nTramos usados por el vehículo ",h,":")
        forall(i,j in nodos|x(i,j,h).sol>=0.99)write(i,"->",j,",")
end-do
end-model
```

Anexo 2: Programación de modelos exactos para flota heterogénea

Modelo de tres índices basado en Tucker- Miller- Zemlin

```
model "Modelo de tres índices basado en Tucker- Miller- Zemlin"
uses "mmxprs";
uses "mmsystem"
declarations
        n:integer
        m: integer
        archivo_datos = "C:/eilon_7_2_heter.dat"
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos=1..n
        vehiculos=1..m
        costo:array(nodos,nodos)of real
        dem: array(nodos) of real ! demanda
        cap:array(vehiculos)of real !capacidad de cada vehiculo
        i1:integer
        x: array(nodos, nodos, vehiculos) of mpvar
        y: array(nodos, vehiculos) of mpvar
        u: array (nodos, vehiculos)of mpvar
        tiempo_max = -200
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(costo(i,j))
forall(k in vehiculos)read(cap(k))
fclose(F_INPUT)
forall(i in nodos)costo(i,i):=9999.9
writeln("costo = ",costo)
!=================MODELO=================
costo_total:=sum (i in nodos, j in nodos, h in vehiculos) costo(i,j)*x(i,j,h)
forall(i in nodos|i>1)res3(i):=sum(h in vehiculos)y(i,h)=1
forall(i in nodos, h in vehiculos|i>1)cot(i,h):=dem(i)<=u(i,h)
forall(i in nodos, h in vehiculos|i>1)cot0(i,h):=u(i,h)<=cap(h)
forall(i in nodos, j in nodos, h in vehiculos|i>1 and j>1 and i<>j and
dem(i)+dem(j)<=cap(h)) cot3(i,j,h):=u(i,h)-u(j,h)+cap(h)*x(i,j,h)<=cap(h)-dem(j)
forall(i in nodos,h in vehiculos)res5(i,h):=sum(j in nodos)x(i,j,h)=y(i,h)
forall(i in nodos,h in vehiculos)res6(i,h):=sum(j in nodos)x(j,i,h)=y(i,h)
forall(h in vehiculos)res7(h):=sum(i in nodos)dem(i)*y(i,h)<=cap(h)
forall(i in nodos, j in nodos,h in vehiculos)x(i,j,h) is_binary
forall(i in nodos, h in vehiculos)y(i,h)is_binary

!===============OPTIMIZACIÓN==============
setparam("XPRS_MAXTIME",tiempo_max)
minimize(costo_total)
```

```
!================RESULTADOS================
writeln("\nSolución obtenida en ",gettime," segundos:\n")
writeln("Costo = ",getobjval)
writeln("\nAsignación de nodos a vehículos:")
forall(i in nodos,h in vehiculos|y(i,h).sol>=0.99)writeln("y(",i,",",h,") = ",y(i,h).sol)
forall(h in vehiculos)do
        writeln("\nTramos usados por el vehículo ",h,":")
        forall(i,j in nodos|x(i,j,h).sol>=0.99)write(i,"->",j,",")
end-do
end-model
```

Modelo de Golden- Assad- Levys- Gheysens basado en Tucker- Miller- Zemlin

```
model "Modelo de Golden- Assad- Gheysens, basado en Tucker- Miller- Zemlin"
uses "mmxprs";
uses "mmsystem"
declarations
        n:integer
        archivo_datos = "C:eilon_7_2_heter.dat"
end-declarations
fopen(archivo_datos,F_INPUT)
        read(n)
        read(m)
declarations
        nodos = 1..n
        vehiculos = 1..m
        costo:array(nodos,nodos)of real
        dem: array(nodos) of real ! demanda
        cap:array(vehiculos)of real !capacidad de cada vehiculo
        cap_usada:array(vehiculos)of real !capacidad usada de cada vehiculo
        i1:integer
        tiempo_max = -200
        x: array(nodos, nodos, vehiculos) of mpvar
        r: array (nodos)of mpvar
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(costo(i,j))
forall(k in vehiculos)read(cap(k))
fclose(F_INPUT)
forall(i in nodos)costo(i,i):=9999.9
capmax:=max(h in vehiculos)cap(h)
writeln(m," vehículos, ",n," puntos")
writeln("capacidades = ",cap)
!================MODELO================
costo_total:=sum (i in nodos, j in nodos, h in vehiculos) costo(i,j)*x(i,j,h)
forall(j in nodos|j>1)res1(j):=sum(h in vehiculos,i in nodos)x(i,j,h)=1
forall(i in nodos,h in vehiculos)res2(i,h):=sum(j in nodos)x(i,j,h)-sum(j in nodos)x(j,i,h)=0
res3:=r(1)=0
forall(i in nodos,j in nodos|j>1)res4(i,j):=r(j)-r(i)>=(dem(j)+capmax)*sum(h in vehiculos)x(i,j,h)-
capmax
forall(j in nodos|j>1)res5(j):=r(j)<=sum(h in vehiculos,i in nodos)cap(h)*x(i,j,h)
forall(h in vehiculos)res6(h):=sum(j in nodos|j>1)x(1,j,h)<=1
forall(i in nodos, j in nodos,h in vehiculos)x(i,j,h) is_binary
!==============OPTIMIZACIÓN==============
```

```
setparam("XPRS_MAXTIME",tiempo_max)
minimize(costo_total)
!================RESULTADOS================
writeln("\nSolución obtenida en ",gettime," segundos:\n")
writeln("Costo = ",getobjval)
forall(h in vehiculos)cap_usada(h):=sum(i,j in nodos)dem(j)*x(i,j,h).sol
forall(h in vehiculos)do
        writeln("\n\nVehículo ",h,", capacidad usada: ",cap_usada(h),", tramos:")
        if(cap_usada(h)>0.01)then
                forall(i,j in nodos|x(i,j,h).sol>=0.99)write(i,"->",j,",")
        end-if
end-do
writeln("\n")
forall(i in nodos)writeln("r(",i,") = ",r(i).sol)
end-model
```

Modelo de Golden- Assad- Gheysens basado en redes

```
model "Model de Golden- Assad- Gheysens, basado en redes "
uses "mmxprs";
uses "mmsystem"
declarations
        n:integer
        m: integer !numero de vehículos
        archivo_datos = "C:eilon_7_2_heter.dat"
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        vehiculos = 1..m
        costo:array(nodos,nodos)of real
        dem: array(nodos) of real
        b: array(nodos) of real
        cap:array(vehiculos)of real
        capmax:real
        cap_usada:array(vehiculos)of real
        t1:real
        i1:integer
        tiempo_max = -200
        x: array(nodos, nodos, vehiculos) of mpvar
        y: array (nodos,nodos)of mpvar
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(costo(i,j))
forall(k in vehiculos)read(cap(k))
fclose(F_INPUT)
forall(i in nodos)costo(i,i):=9999.9
capmax:=sum(k in vehiculos)cap(k)
b(1):=sum(i in nodos|i>1)dem(i)
forall(i in nodos|i>1)b(i):=-dem(i)
writeln(m," vehículos, ",n," puntos")
writeln("capacidades = ",cap)
writeln("b = ",b)
```

```
!=================MODELO================
costo_total:=sum (i in nodos, j in nodos, h in vehiculos) costo(i,j)*x(i,j,h)
forall(j in nodos|j>1)res1(j):=sum(h in vehiculos,i in nodos)x(i,j,h)=1
forall(i in nodos,h in vehiculos)res2(i,h):=sum(j in nodos)x(i,j,h)-sum(j in nodos)x(j,i,h)=0
forall(i in nodos|i>1)res3(i):=sum(j in nodos)y(i,j)-sum(j in nodos)y(j,i)=b(i)
forall(j in nodos,k in vehiculos|j>1)res4(j,k):=y(1,j)-cap(k)<=(capmax-cap(k))*(1-x(1,j,k))
forall(i,j in nodos|i<>j)res5(i,j):=y(i,j)<=sum(k in vehiculos)cap(k)*x(i,j,k)
forall(k in vehiculos)res6(k):=sum(j in nodos)x(1,j,k)<=1
forall(i in nodos, j in nodos,h in vehiculos)x(i,j,h) is_binary
!===============OPTIMIZACIÓN==============
setparam("XPRS_MAXTIME",tiempo_max)
minimize(costo_total)
!================RESULTADOS================
writeln("\nSolución obtenida en ",gettime," segundos:\n")
writeln("Costo = ",getobjval)
forall(h in vehiculos)cap_usada(h):=sum(i,j in nodos)dem(j)*x(i,j,h).sol
forall(h in vehiculos)do
        writeln("\n\nVehículo ",h,", capacidad usada: ",cap_usada(h),", tramos:")
        if(cap_usada(h)>0.01)then
                forall(i,j in nodos|x(i,j,h).sol>=0.99)write(i,"->",j,",")
        end-if
end-do
writeln("\n\nVariables de flujo:\n\n")
forall(i,j in nodos|y(i,j).sol>=0.01)writeln("y(",i,",",j,") = ",y(i,j).sol)
writeln("\nTiempo = ",gettime-t1," seg.")
end-model
```

Modelo de Christofides- Mingozzi- Toth

```
model "Modelo de Christofides-Mingozzi-Toth"
uses "mmxprs";
uses "mmsystem"
declarations
        n:integer
        m: integer
        archivo_datos = "C:/eilon_7_2_heter.dat"
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        vehiculos = 1..m
        costo:array(nodos,nodos)of real
        dem: array(nodos) of real
        cap:array(vehiculos)of real
        cap_usada:array(vehiculos)of real
        t1:real
        i1:integer
        tiempo_max = -200
        x: array(nodos, nodos, vehiculos) of mpvar
        r: array (nodos)of mpvar
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(costo(i,j))
```

```
forall(k in vehiculos)read(cap(k))
fclose(F_INPUT)
forall(i in nodos)costo(i,i):=9999.9
capmax:=max(h in vehiculos)cap(h)
writeln(m," vehículos, ",n," puntos")
writeln("capacidades = ",cap)
!=================MODELO================
costo_total:=sum (i in nodos, j in nodos, h in vehiculos) costo(i,j)*x(i,j,h)
forall(j in nodos|j>1)res1(j):=sum(h in vehiculos,i in nodos)x(i,j,h)=1
forall(i in nodos,h in vehiculos)res2(i,h):=sum(j in nodos)x(i,j,h)-sum(j in nodos)x(j,i,h)=0
forall(i in nodos,j in nodos|j>1)res4(i,j):=r(i)-r(j)+n*sum(h in vehiculos)x(i,j,h)<=n-1
forall(h in vehiculos)res5(h):=sum(i in nodos|i>1)dem(i)*sum(j in nodos)x(i,j,h)<=cap(h)
forall(h in vehiculos)res6(h):=sum(j in nodos|j>1)x(1,j,h)<=1
forall(i in nodos, j in nodos,h in vehiculos)x(i,j,h) is_binary
t1:=gettime
!===============OPTIMIZACIÓN===============
setparam("XPRS_MAXTIME",tiempo_max)
minimize(costo_total)

writeln("\nSolución obtenida en ",gettime," segundos:\n")
writeln("Costo = ",getobjval)
!================RESULTADOS================
forall(h in vehiculos)cap_usada(h):=sum(i,j in nodos)dem(j)*x(i,j,h).sol
forall(h in vehiculos)do
        writeln("\n\nVehículo ",h,", capacidad usada: ",cap_usada(h),", tramos:")
        if(cap_usada(h)>0.01)then
                forall(i,j in nodos|x(i,j,h).sol>=0.99)write(i,"->",j,",")
        end-if
end-do
writeln("\n")
forall(i in nodos)writeln("r(",i,") = ",r(i).sol)
writeln("\nTiempo = ",gettime-t1," seg.")
end-model
```

Modelo multiproducto

```
model "Modelo multiproducto"
uses "mmxprs";
uses "mmsystem"
declarations
        n:integer
        m: integer !numero de vehículos
        archivo_datos = "C:eilon_7_2_heter.dat"
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        vehiculos = 1..m
        costo:array(nodos,nodos)of real
        dem: array(nodos) of real
        b: array(nodos) of real
        cap:array(vehiculos)of real
        capmax:real
        cap_usada:array(vehiculos)of real
        t1:real
         i1:integer
        tiempo_max = -200
```

```
        x: array(nodos, nodos, vehiculos) of mpvar
        y: array (nodos,nodos,vehiculos)of mpvar
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(costo(i,j))
forall(k in vehiculos)read(cap(k))
fclose(F_INPUT)
forall(i in nodos)costo(i,i):=9999.9
capmax:=sum(k in vehiculos)cap(k)
b(1):=sum(i in nodos|i>1)dem(i)
forall(i in nodos|i>1)b(i):=-dem(i)
writeln(m," vehículos, ",n," puntos")
writeln("capacidades = ",cap)
writeln("b = ",b)
!=================MODELO================
costo_total:=sum (i in nodos, j in nodos, h in vehiculos) costo(i,j)*x(i,j,h)
forall(j in nodos|j>1)res1(j):=sum(h in vehiculos,i in nodos)x(i,j,h)=1
forall(i in nodos,h in vehiculos)res2(i,h):=sum(j in nodos)x(i,j,h)-sum(j in nodos)x(j,i,h)=0
forall(i in nodos,k in vehiculos|i>1)res3(i,k):=sum(j in nodos)y(i,j,k)-sum(j in nodos)y(j,i,k)= -
dem(i)*sum(j in nodos)x(i,j,k)
forall(i,j in nodos,k in vehiculos|i<>j)res5(i,j,k):=y(i,j,k)<=cap(k)*x(i,j,k)
forall(k in vehiculos)res6(k):=sum(j in nodos)x(1,j,k)<=1
forall(i in nodos, j in nodos,h in vehiculos)x(i,j,h) is_binary
!===============OPTIMIZACIÓN==============
setparam("XPRS_MAXTIME",tiempo_max)
minimize(costo_total)
!================RESULTADOS================
writeln("\nSolución obtenida en ",gettime," segundos:\n")
writeln("Costo = ",getobjval)
forall(h in vehiculos)cap_usada(h):=sum(i,j in nodos)dem(j)*x(i,j,h).sol
forall(h in vehiculos)do
        writeln("\n\nVehículo ",h,", capacidad usada: ",cap_usada(h),", tramos:")
        if(cap_usada(h)>0.01)then
                forall(i,j in nodos|x(i,j,h).sol>=0.99)write(i,"->",j,",")
        end-if
end-do
writeln("\nTiempo = ",gettime-t1," seg.")
end-model
```

Anexo 3: Programación de heurísticas

Algoritmo de Ahorros de Ckarke & Wright

```
model " Heuristica de Clarke & Wright"
uses "mmxprs"
uses "mmsystem"
parameters
        archivo_datos =  "C:eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        dist:array(nodos,nodos) of integer
        dem:array(nodos)of integer
        cap:integer
        rutas = 1..n-1
        nnr:array(rutas)of integer
        indr:array(nodos,rutas)of integer
        ruta_activa:array(rutas)of integer
        carga:array(rutas)of integer
        ind:array(nodos,rutas)of integer
        nra:integer
        sig:array(nodos,rutas)of integer
        siga:array(nodos,rutas)of integer
        ini:array(rutas) of integer
        fin:array(rutas) of integer
        sav:array(nodos,nodos)of integer
        nsav:integer
        savord:array(range)of integer
        indi:array(range)of integer
        indj:array(range)of integer
        marcado:array(nodos,nodos)of integer
        dist_ruta:array(rutas)of integer
        distotal, daux:integer
        i1, final, iter, ruta, ic, jact, is, js, t1:integer
        inicial = 1
        tol = 0.001
        tini:real
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================DATOS==================
writeln("Problema de routing con capacidades  CVRP, \nn = ",n,", m = ",m)
writeln("Datos: ",archivo_datos)
writeln("\Heuristica de Clarke-Wright")
```

```
writeln("\nDemandas: ")
!forall(i in nodos)writeln(dem(i))
writeln("Demanda total: ",dem_total)
tini:=gettime
writeln("\nMatriz de distancias:\n")
forall(i in nodos)do
  writeln
  forall(j in nodos)write(dist(i,j),"\t")
end-do
writeln("\nCapacidad = ",cap)
writeln("Mínimo número de vehículos: ",ceil(dem_total/cap))
writeln("Número de vehículos: ",m)
!==========SOLUCIÓN INICIAL: n-1 rutas=========
nra:=0
dsol:=0
forall(i in nodos|i>1)do
        nra:=nra+1
        nnr(nra):=1
        sig(i,nra):=1
        sig(1,nra):=i
        ruta_activa(nra):=1
        carga(nra):=dem(i)
        ini(nra):=i
        fin(nra):=i
        dist_ruta(nra):=dist(1,i)+dist(i,1)
        dsol:=dsol+dist(1,i)+dist(i,1)
end-do
writeln("\nSolución inicial, dist = ",dsol)
!======CÁLCULO DE LA MATRIZ DE AHORROS=====
forall(i,j in nodos|i>1 and j>1)sav(i,j):=dist(i,1)+dist(1,j)-dist(i,j)
writeln("\nMatriz de ahorros:\n")
forall(j in nodos|j>1)write(j,"\t")
writeln
forall(i in nodos|i>1 )do
        write("\n",i,"\t")
        forall(j in nodos|j>1)write(sav(i,j),"\t")
end-do
!===LISTA ORDENADA DE AHORROS POSITIVOS===
nsav:=0
final:=0
while(final=0)do
        nsav:=nsav+1
        daux:=-999
        forall(i,j in nodos|i>1 and j>1 and marcado(i,j)=0)do
                if(sav(i,j)>daux)then
                        daux:=sav(i,j)
                        ik:=i
                        jk:=j
                end-if
        end-do
        marcado(ik,jk):=1
        savord(nsav):=daux
        indi(nsav):=ik
        indj(nsav):=jk
        if(daux<0)then final:=1;end-if
end-do
nsav:=nsav-1
writeln("\nSavings ordenados:\n")
writeln("t\tsav\ti\tj\n")
```

```
forall(t in 1..nsav)writeln(t,"\t",savord(t),"\t",indi(t),"\t",indj(t))
!=============BUCLE PRINCIPAL=============
forall(t in 1..nsav)do
        is:=indi(t)
        js:=indj(t)
        forall(k1, k2 in rutas|k1<>k2 and ruta_activa(k1)=1 and ruta_activa(k2)=1)do
                if(is=fin(k2) and js=ini(k1) and carga(k1)+carga(k2)<=cap)then
                        writeln("mezcla de las rutas ",k1," y ",k2,", is = ",is,", js = ",js,", saving =
                        ",savord(t))
                        ruta_activa(k1):=0
                        sig(is,k2):=js
                        jact:=js
                        while(jact<>fin(k1))do
                                sig(jact,k2):=sig(jact,k1)
                                jact:=sig(jact,k1)
                        end-do
                        fin(k2):=fin(k1)
                        sig(fin(k2),k2):=1
                        carga(k2):=carga(k1)+carga(k2)
                        nnr(k2):=nnr(k1)+nnr(k2)
                        nra:=nra-1
                        break
                end-if
        end-do
end-do
!================RESULTADOS================
writeln("\nSolución final de la heurística de Clarke and Wright:")
writeln("Número de rutas: ",nra)
nra:=0
distotal:=0  ! distancia total de la ruta
forall(k1 in rutas|ruta_activa(k1)=1)do
        nra:=nra+1
        write("\nruta ",k1,", carga: ",carga(k1),", ini = ",ini(k1),", fin = ",fin(k1),", nrecorrido: ")
        t1:=1;indr(t1,k1):=1
        write(1)
        jact:=ini(k1)
        t1:=t1+1
        indr(t1,k1):=jact
        write("-",jact)
        distotal:=distotal+dist(1,ini(k1))
        while(jact<>fin(k1))do
                distotal:=distotal+dist(jact, sig(jact,k1))
                jact:=sig(jact,k1)
                t1:=t1+1
                indr(t1,k1):=jact
                write("-",jact)
        end-do
        write("-",1)
        distotal:=distotal+dist(fin(k1),1)
        write(", nnr = ",nnr(k1)+1,", t1 = ",t1)
        nnr(k1):=t1
end-do
writeln("\n\nIndices de cada ruta:")
forall(k in rutas|ruta_activa(k)=1)do
        write("\nruta : ",k," => ")
        forall(t in 1..nnr(k))write(indr(t,k),",")
end-do
writeln("\n\nDistancia total = ",distotal)
writeln("Tiempo total = ",gettime-tini)
```

```
end-model
```

Modificación del Algoritmo de Ahorros de Clarke & Wright mediante un parámetro

```
model " Modificación 1 parámetro heuristica de Clarke & Wright"
uses "mmxprs"
uses "mmsystem"
parameters
        archivo_datos =  "C:eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        dist:array(nodos,nodos) of integer
        dem:array(nodos)of integer
        cap:integer
        rutas = 1..n-1
        nnr:array(rutas)of integer
        indr:array(nodos,rutas)of integer
        ruta_activa:array(rutas)of integer
        carga:array(rutas)of integer
        ind:array(nodos,rutas)of integer
        nra:integer
        sig:array(nodos,rutas)of integer
        siga:array(nodos,rutas)of integer
        ini:array(rutas) of integer
        fin:array(rutas) of integer
        sav:array(nodos,nodos)of real
        nsav:integer
        savord:array(range)of real
        indi:array(range)of integer
        indj:array(range)of integer
        marcado:array(nodos,nodos)of integer
        dist_ruta:array(rutas)of integer
        distotal:integer
        daux:real
        i1, final, iter, ruta, ic, jact, is, js, t1:integer
        inicial = 1
        tol = 0.001
        tini:real
        lambda:real
        lambda_min = 0.1
        lambda_max = 2.
        lambda_mejor:real
        N = 20
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
```

```
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================DATOS==================
writeln("Problema de routing con capacidades  CVRP, \nn = ",n,", m = ",m)
writeln("Datos: ",archivo_datos)
writeln("\nDemandas: ")
!forall(i in nodos)writeln(dem(i))
writeln("Demanda total: ",dem_total)
tini:=gettime
writeln("\nMatriz de distancias:\n")
forall(i in nodos)do
  writeln
  forall(j in nodos)write(dist(i,j),"\t")
end-do
writeln("\nCapacidad = ",cap)
writeln("Mínimo número de vehículos: ",ceil(dem_total/cap))
writeln("Número de vehículos: ",m)
dist_mejor:=9999999
forall(krep in 1..N)do
        lambda:=lambda_min+(krep-1)*(lambda_max-lambda_min)/(N-1)
!==========SOLUCIÓN INICIAL: n-1 rutas=========
nra:=0
dsol:=0
forall(i in nodos|i>1)do
        nra:=nra+1
        nnr(nra):=1
        sig(i,nra):=1
        sig(1,nra):=i
        ruta_activa(nra):=1
        carga(nra):=dem(i)
        ini(nra):=i
        fin(nra):=i
        dist_ruta(nra):=dist(1,i)+dist(i,1)
        dsol:=dsol+dist(1,i)+dist(i,1)
end-do
writeln("\nSolución inicial, dist = ",dsol)
!======CÁLCULO DE LA MATRIZ DE AHORROS=====
forall(i,j in nodos|i>1 and j>1)sav(i,j):=dist(i,1)+dist(1,j)-dist(i,j) -lambda*dist(i,j)
writeln("\nMatriz de ahorros:\n")
forall(j in nodos|j>1)write(j,"\t")
writeln
forall(i in nodos|i>1 )do
        write("\n",i,"\t")
        forall(j in nodos|j>1)write(sav(i,j),"\t")
end-do
!===LISTA ORDENADA DE AHORROS POSITIVOS===
nsav:=0
final:=0
while(final=0)do
        nsav:=nsav+1
        daux:=-999
        forall(i,j in nodos|i>1 and j>1 and marcado(i,j)=0)do
                if(sav(i,j)>daux)then
                        daux:=sav(i,j)
                        ik:=i
                        jk:=j
                end-if
        end-do
        marcado(ik,jk):=1
```

```
        savord(nsav):=daux
        indi(nsav):=ik
        indj(nsav):=jk
        if(daux<0)then final:=1;end-if
end-do
nsav:=nsav-1
writeln("\nSavings ordenados:\n")
writeln("t\tsav\ti\tj\n")
forall(t in 1..nsav)writeln(t,"\t",savord(t),"\t",indi(t),"\t",indj(t))
!=============BUCLE PRINCIPAL=============
forall(t in 1..nsav)do
        is:=indi(t)
        js:=indj(t)
        forall(k1, k2 in rutas|k1<>k2 and ruta_activa(k1)=1 and ruta_activa(k2)=1)do
                if(is=fin(k2) and js=ini(k1) and carga(k1)+carga(k2)<=cap)then
                        writeln("mezcla de las rutas ",k1," y ",k2,", is = ",is,", js = ",js,", saving =
                        ",savord(t))
                        ruta_activa(k1):=0
                        sig(is,k2):=js
                        jact:=js
                        while(jact<>fin(k1))do
                                sig(jact,k2):=sig(jact,k1)
                                jact:=sig(jact,k1)
                        end-do
                        fin(k2):=fin(k1)
                        sig(fin(k2),k2):=1
                        carga(k2):=carga(k1)+carga(k2)
                        nnr(k2):=nnr(k1)+nnr(k2)
                        nra:=nra-1
                        break
                end-if
        end-do
end-do
!================RESULTADOS================
writeln("\nSolución final:")
writeln("Número de rutas: ",nra)
nra:=0
distotal:=0  ! distancia total de la ruta
forall(k1 in rutas|ruta_activa(k1)=1)do
        nra:=nra+1
        write("\nruta ",k1,", carga: ",carga(k1),", ini = ",ini(k1),", fin = ",fin(k1),", nrecorrido: ")
        t1:=1;indr(t1,k1):=1
        write(1)
        jact:=ini(k1)
        t1:=t1+1
        indr(t1,k1):=jact
        write("-",jact)
        distotal:=distotal+dist(1,ini(k1))
        while(jact<>fin(k1))do
                distotal:=distotal+dist(jact, sig(jact,k1))
                jact:=sig(jact,k1)
                t1:=t1+1
                indr(t1,k1):=jact
                write("-",jact)
        end-do
        write("-",1)
        distotal:=distotal+dist(fin(k1),1)
        write(", nnr = ",nnr(k1)+1,", t1 = ",t1)
        nnr(k1):=t1
```

```
end-do
writeln("\n\nIndices de cada ruta:")
forall(k in rutas|ruta_activa(k)=1)do
        write("\nruta : ",k," => ")
        forall(t in 1..nnr(k))write(indr(t,k),",")
end-do
writeln("\n\nDistancia total = ",distotal)
writeln("Tiempo total = ",gettime-tini)
end-model
```

Modificación del Algoritmo de Ahorros de Ckarke & Wright mediante dos parámetros

```
model " Modificación 2 parámetros heuristica de Clarke & Wright"
uses "mmxprs"
uses "mmsystem"
parameters
        archivo_datos =  "C:eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        dist:array(nodos,nodos) of integer
        dem:array(nodos)of integer
        cap:integer
        rutas = 1..n-1
        nnr:array(rutas)of integer
        indr:array(nodos,rutas)of integer
        ruta_activa:array(rutas)of integer
        carga:array(rutas)of integer
        ind:array(nodos,rutas)of integer
        nra:integer
        sig:array(nodos,rutas)of integer
        siga:array(nodos,rutas)of integer
        ini:array(rutas) of integer
        fin:array(rutas) of integer
        sav:array(nodos,nodos)of real
        nsav:integer
        savord:array(range)of real
        indi:array(range)of integer
        indj:array(range)of integer
        marcado:array(nodos,nodos)of integer
        dist_ruta:array(rutas)of integer
        distotal:integer
        daux:real
        i1, final, iter, ruta, ic, jact, is, js, t1:integer
        inicial = 1
        tol = 0.001
        tini:real
        lambda, mu: real

        lambda_min = 0.4
        lambda_max = 1.8
```

```
            mu_min = 0.6
            mu_max = 2.
            lambda_mejor, mu_mejor:real
            N1 = 10
            N2 = 8
end-declarations
forall(i in nodos)do
            read(i1)
            read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================DATOS==================
writeln("Problema de routing con capacidades  CVRP, \nn = ",n,", m = ",m)
writeln("Datos: ",archivo_datos)
writeln("\nDemandas: ")
!forall(i in nodos)writeln(dem(i))
writeln("Demanda total: ",dem_total)
tini:=gettime
writeln("\nMatriz de distancias:\n")
forall(i in nodos)do
  writeln
  forall(j in nodos)write(dist(i,j),"\t")
end-do
writeln("\nCapacidad = ",cap)
writeln("Mínimo número de vehículos: ",ceil(dem_total/cap))
writeln("Número de vehículos: ",m)
dist_mejor:=9999999
forall(r1 in 1..N1,r2 in 1..N2)do
   lambda:=lambda_min+(r1-1)*(lambda_max-lambda_min)/(N1-1)
   mu:=mu_min+(r2-1)*(mu_max-mu_min)/(N2-1)
!==========SOLUCIÓN INICIAL: n-1 rutas=========
nra:=0
dsol:=0
forall(i in nodos|i>1)do
            nra:=nra+1
            nnr(nra):=1
            sig(i,nra):=1
            sig(1,nra):=i
            ruta_activa(nra):=1
            carga(nra):=dem(i)
            ini(nra):=i
            fin(nra):=i
            dist_ruta(nra):=dist(1,i)+dist(i,1)
            dsol:=dsol+dist(1,i)+dist(i,1)
end-do
writeln("\nSolución inicial, dist = ",dsol)
!======CÁLCULO DE LA MATRIZ DE AHORROS=====
forall(i,j in nodos|i>1 and j>1)sav(i,j):=dist(i,1)+dist(1,j)-dist(i,j) -lambda*dist(i,j)
+mu*abs(dist(i,1)+dist(1,j))
writeln("\nMatriz de ahorros:\n")
forall(j in nodos|j>1)write(j,"\t")
writeln
forall(i in nodos|i>1 )do
            write("\n",i,"\t")
            forall(j in nodos|j>1)write(sav(i,j),"\t")
end-do
```

```
!===LISTA ORDENADA DE AHORROS POSITIVOS===
nsav:=0
final:=0
while(final=0)do
        nsav:=nsav+1
        daux:=-999
        forall(i,j in nodos|i>1 and j>1 and marcado(i,j)=0)do
                if(sav(i,j)>daux)then
                        daux:=sav(i,j)
                        ik:=i
                        jk:=j
                end-if
        end-do
        marcado(ik,jk):=1
        savord(nsav):=daux
        indi(nsav):=ik
        indj(nsav):=jk
        if(daux<0)then final:=1;end-if
end-do
nsav:=nsav-1
writeln("\nSavings ordenados:\n")
writeln("t\tsav\ti\tj\n")
forall(t in 1..nsav)writeln(t,"\t",savord(t),"\t",indi(t),"\t",indj(t))
!=============BUCLE PRINCIPAL=============
forall(t in 1..nsav)do
        is:=indi(t)
        js:=indj(t)
        forall(k1, k2 in rutas|k1<>k2 and ruta_activa(k1)=1 and ruta_activa(k2)=1)do
                if(is=fin(k2) and js=ini(k1) and carga(k1)+carga(k2)<=cap)then
                        writeln("mezcla de las rutas ",k1," y ",k2,", is = ",is,", js = ",js,", saving =
                        ",savord(t))
                        ruta_activa(k1):=0
                        sig(is,k2):=js
                        jact:=js
                        while(jact<>fin(k1))do
                                sig(jact,k2):=sig(jact,k1)
                                jact:=sig(jact,k1)
                        end-do
                        fin(k2):=fin(k1)
                        sig(fin(k2),k2):=1
                        carga(k2):=carga(k1)+carga(k2)
                        nnr(k2):=nnr(k1)+nnr(k2)
                        nra:=nra-1
                        break
                end-if
        end-do
end-do
!================RESULTADOS================
writeln("\nSolución final:")
writeln("Número de rutas: ",nra)
nra:=0
distotal:=0  ! distancia total de la ruta
forall(k1 in rutas|ruta_activa(k1)=1)do
        nra:=nra+1
        write("\nruta ",k1,", carga: ",carga(k1),", ini = ",ini(k1),", fin = ",fin(k1),", nrecorrido: ")
        t1:=1;indr(t1,k1):=1
        write(1)
        jact:=ini(k1)
        t1:=t1+1
```

```
        indr(t1,k1):=jact
        write("-",jact)
        distotal:=distotal+dist(1,ini(k1))
        while(jact<>fin(k1))do
                distotal:=distotal+dist(jact, sig(jact,k1))
                jact:=sig(jact,k1)
                t1:=t1+1
                indr(t1,k1):=jact
                write("-",jact)
        end-do
        write("-",1)
        distotal:=distotal+dist(fin(k1),1)
        write(", nnr = ",nnr(k1)+1,", t1 = ",t1)
        nnr(k1):=t1
end-do
writeln("\n\nIndices de cada ruta:")
forall(k in rutas|ruta_activa(k)=1)do
        write("\nruta : ",k," => ")
        forall(t in 1..nnr(k))write(indr(t,k),",")
end-do
writeln("\n\nDistancia total = ",distotal)
writeln("Tiempo total = ",gettime-tini)
end-model
```

Modificación del Algoritmo de Ahorros de Ckarke & Wright mediante tres parámetros

```
model " Modificación 2 parámetro heuristica de Clarke & Wright"
uses "mmxprs"
uses "mmsystem"
parameters
        archivo_datos =  "C:eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        dist:array(nodos,nodos) of integer
        dem:array(nodos)of integer
        cap:integer
        rutas = 1..n-1
        nnr:array(rutas)of integer
        indr:array(nodos,rutas)of integer
        ruta_activa:array(rutas)of integer
        carga:array(rutas)of integer
        ind:array(nodos,rutas)of integer
        nra:integer
        sig:array(nodos,rutas)of integer
        siga:array(nodos,rutas)of integer
        ini:array(rutas) of integer
        fin:array(rutas) of integer
        sav:array(nodos,nodos)of real
        nsav:integer
        savord:array(range)of real
```

```
        indi:array(range)of integer
        indj:array(range)of integer
        marcado:array(nodos,nodos)of integer
        dist_ruta:array(rutas)of integer
        distotal:integer
        daux:real
        i1, final, iter, ruta, ic, jact, is, js, t1:integer
        inicial = 1
        tol = 0.001
        tini:real
        lambda, mu, nu: real
        lambda_min = 0.4
        lambda_max = 1.8
        mu_min = 0.6
        mu_max = 2.
        nu_min = 0.6
        nu_max = 1.8
        lambda_mejor, mu_mejor, nu_mejor:real
        N1 = 10
        N2 = 8
        N3 = 8
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================DATOS==================
writeln("Problema de routing con capacidades  CVRP, \nn = ",n,", m = ",m)
writeln("Datos: ",archivo_datos)
writeln("\nDemandas: ")
!forall(i in nodos)writeln(dem(i))
writeln("Demanda total: ",dem_total)
tini:=gettime
writeln("\nMatriz de distancias:\n")
forall(i in nodos)do
  writeln
  forall(j in nodos)write(dist(i,j),"\t")
end-do
writeln("\nCapacidad = ",cap)
writeln("Mínimo número de vehículos: ",ceil(dem_total/cap))
writeln("Número de vehículos: ",m)
dist_mejor:=9999999
forall(r1 in 1..N1,r2 in 1..N2,r3 in 1..N3)do
   lambda:=lambda_min+(r1-1)*(lambda_max-lambda_min)/(N1-1)
   mu:=mu_min+(r2-1)*(mu_max-mu_min)/(N2-1)
   nu:=nu_min+(r3-1)*(nu_max-nu_min)/(N3-1)
!==========SOLUCIÓN INICIAL: n-1 rutas=========
nra:=0
dsol:=0
forall(i in nodos|i>1)do
        nra:=nra+1
        nnr(nra):=1
        sig(i,nra):=1
        sig(1,nra):=i
        ruta_activa(nra):=1
```

```
        carga(nra):=dem(i)
        ini(nra):=i
        fin(nra):=i
        dist_ruta(nra):=dist(1,i)+dist(i,1)
        dsol:=dsol+dist(1,i)+dist(i,1)
end-do
writeln("\nSolución inicial, dist = ",dsol)
!======CÁLCULO DE LA MATRIZ DE AHORROS=====
forall(i,j in nodos|i>1 and j>1)sav(i,j):=dist(i,1)+dist(1,j)-dist(i,j) -lambda*dist(i,j)
+nu*(dem(i)+dem(j))/dem_media+mu*abs(dist(i,1)+dist(1,j))
writeln("\nMatriz de ahorros:\n")
forall(j in nodos|j>1)write(j,"\t")
writeln
forall(i in nodos|i>1 )do
        write("\n",i,"\t")
        forall(j in nodos|j>1)write(sav(i,j),"\t")
end-do
!===LISTA ORDENADA DE AHORROS POSITIVOS===
nsav:=0
final:=0
while(final=0)do
        nsav:=nsav+1
        daux:=-999
        forall(i,j in nodos|i>1 and j>1 and marcado(i,j)=0)do
                if(sav(i,j)>daux)then
                        daux:=sav(i,j)
                        ik:=i
                        jk:=j
                end-if
        end-do
        marcado(ik,jk):=1
        savord(nsav):=daux
        indi(nsav):=ik
        indj(nsav):=jk
        if(daux<0)then final:=1;end-if
end-do
nsav:=nsav-1
writeln("\nSavings ordenados:\n")
writeln("t\tsav\ti\tj\n")
forall(t in 1..nsav)writeln(t,"\t",savord(t),"\t",indi(t),"\t",indj(t))
!=============BUCLE PRINCIPAL=============
forall(t in 1..nsav)do
        is:=indi(t)
        js:=indj(t)
        forall(k1, k2 in rutas|k1<>k2 and ruta_activa(k1)=1 and ruta_activa(k2)=1)do
                if(is=fin(k2) and js=ini(k1) and carga(k1)+carga(k2)<=cap)then
                        writeln("mezcla de las rutas ",k1," y ",k2,", is = ",is,", js = ",js,", saving =
                        ",savord(t))
                        ruta_activa(k1):=0
                        sig(is,k2):=js
                        jact:=js
                        while(jact<>fin(k1))do
                                sig(jact,k2):=sig(jact,k1)
                                jact:=sig(jact,k1)
                        end-do
                        fin(k2):=fin(k1)
                        sig(fin(k2),k2):=1
                        carga(k2):=carga(k1)+carga(k2)
                        nnr(k2):=nnr(k1)+nnr(k2)
```

```
                        nra:=nra-1
                        break
                end-if
        end-do
end-do
!================RESULTADOS================
writeln("\nSolución final:")
writeln("Número de rutas: ",nra)
nra:=0
distotal:=0  ! distancia total de la ruta
forall(k1 in rutas|ruta_activa(k1)=1)do
        nra:=nra+1
        write("\nruta ",k1,", carga: ",carga(k1),", ini = ",ini(k1),", fin = ",fin(k1),", nrecorrido: ")
        t1:=1;indr(t1,k1):=1
        write(1)
        jact:=ini(k1)
        t1:=t1+1
        indr(t1,k1):=jact
        write("-",jact)
        distotal:=distotal+dist(1,ini(k1))
        while(jact<>fin(k1))do
                distotal:=distotal+dist(jact, sig(jact,k1))
                jact:=sig(jact,k1)
                t1:=t1+1
                indr(t1,k1):=jact
                write("-",jact)
        end-do
        write("-",1)
        distotal:=distotal+dist(fin(k1),1)
        write(", nnr = ",nnr(k1)+1,", t1 = ",t1)
        nnr(k1):=t1
end-do
writeln("\n\nIndices de cada ruta:")
forall(k in rutas|ruta_activa(k)=1)do
        write("\nruta : ",k," => ")
        forall(t in 1..nnr(k))write(indr(t,k),",")
end-do
writeln("\n\nDistancia total = ",distotal)
writeln("Tiempo total = ",gettime-tini)
end-model
```

Heurística de Localización de Bramel y Simchi- Levi

```
model "El Problema TSP: Restricciones de Tucker-Miller- Zemlin"
uses "mmxprs";
uses "mmsystem"
uses "mmjobs"
declarations
        nr:integer
        n:integer
end-declarations
initializations from "shmem:datos"
        n nr
end-initializations
declarations
        nodos=1..n
        nodos_ruta=1..nr+1
        dist2:array(nodos_ruta,nodos_ruta) of integer
```

```
            dist:array(nodos,nodos) of integer
            indr:array(nodos)of integer
            dist_ruta:real
            x:array(nodos_ruta,nodos_ruta) of mpvar
            u:array(nodos_ruta) of mpvar
            tiempo_calculo = -200
            i1, iter:integer
            inicial = nr+1
end-declarations
!=================DATOS==================
initializations from "shmem:datos"
  indr dist
end-initializations!)
indr(nr+1):=1
forall(k,j in nodos_ruta|k<nr+1 and j<nr+1)dist2(k,j):=dist(indr(k),indr(j))
forall(j in nodos_ruta|j<nr+1)do
  dist2(j,nr+1):=dist(indr(j),1)
  dist2(nr+1,j):=dist(1,indr(j))
end-do
dist2(nr+1,nr+1):=999
writeln("\n***Problema TSP, n = ",nr+1,", nodos: \n")
forall(j in nodos_ruta)write(indr(j),", ")
writeln("Matriz de distancias:\n")
forall(i in nodos_ruta)do
 writeln
 forall(j in nodos_ruta)write(dist2(i,j),"\t")
end-do
!=================MODELO=================
objetivo:=sum(i,j in nodos_ruta|i<>j)dist2(i,j)*x(i,j)
forall(i,j in nodos_ruta)x(i,j)is_binary
forall(i in nodos_ruta)res_1(i):=sum(j in nodos_ruta|i<>j)x(i,j)=1
forall(j in nodos_ruta)res_2(j):=sum(i in nodos_ruta|i<>j)x(i,j)=1
forall(i,j in nodos_ruta|i<nr+1 and j<nr+1 and i<>j)res_3(i,j):=u(i)-u(j)+(nr+1)*x(i,j)<=nr
!================OPTIMIZACIÓN==============
starttime:=gettime
setparam("XPRS_MAXTIME",tiempo_calculo)
minimize(objetivo)
!================RESULTADOS================
nvar:= getparam("XPRS_cols")
nres:= getparam("XPRS_rows")
writeln("\nNumero de variables: ",nvar,", numero de restricciones: ",nres)
writeln("Valor objetivo :",getobjval)
mip_status:= getparam("XPRS_mipstatus")
mejor_cota:= getparam("XPRS_bestbound")
nodos_ruta_BB:=getparam("XPRS_nodes")
if(getobjval<>0)then
    hueco_final:= 100*(getobjval-mejor_cota)/getobjval
end-if
writeln("\nSolucion final")
writeln("El valor de la función objetivo es: ",getobjval)
writeln("mip_status = ",mip_status)
writeln("Hueco final= ",strfmt(hueco_final,6,2))
writeln("nodos_ruta examinados = ",nodos_ruta_BB)
writeln("Tiempo en segundos = ",gettime-starttime)
dist_ruta:=getobjval
initializations to "raw:noindex"
            dist_ruta as "shmem:dist_ruta"
end-initializations
writeln
```

```
writeln("Recorrido óptimo: distancia = ",getobjval,"\n")
i1:=inicial
iter:=1
write(indr(i1))
while (iter<nr+1)do
        forall(j in nodos_ruta|j<>i1)do
                if(x(i1,j).sol>=0.999)then
                        i1:=j
                        iter:=iter+1
                        write("-",indr(i1))
                end-if
        end-do
end-do
writeln
end-model
!=========================================
model " El Problema concentrador de Bramel-Simchi Levi para CVRP"
uses "mmxprs"
uses "mmsystem"
uses "mmjobs"
parameters
        archivo_datos="C://eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos=1..n
        dist:array(nodos,nodos) of integer
        dem:array(nodos)of integer
        cap:integer
        rutas=1..m
        npc:array(rutas)of integer
        ind:array(rutas,nodos)of integer
        indr:array(nodos)of integer
        dist_ruta, dist_total:real
        puntos=2..n
        c:array(puntos,puntos)of integer
        v:array(puntos)of integer
        x:array(puntos,puntos) of mpvar
        z:array(puntos) of mpvar
        tiempo_calculo = -200
        i1, final, iter, ruta, ic, nr:integer
        inicial = 1
        mod_tsp: Model
        salida_errores_tsp:string
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
```

```
forall(i in puntos,j in puntos)c(i,j):=dist(1,i)+dist(i,j)-dist(1,j)
forall(j in puntos)v(j):=2*dist(1,j)
forall(j in puntos)c(j,j):=0
!=================DATOS==================
writeln("Problema de routing con capacidades, n = ",n,", m = ",m)
writeln("\Problema concentrador de Bramel Simchi-Levi")
!forall(i in nodos)writeln(dem(i))
writeln("Demanda total: ",dem_total)
writeln("\nMatriz de distancias:\n")
forall(i in nodos)do
  writeln
  forall(j in nodos)write(dist(i,j),"\t")
end-do
writeln("Capacidad = ",cap)
writeln("Mínimo número de vehículos: ",ceil(dem_total/cap))
writeln("Número de vehículos: ",m)
!=================MODELO=================
forall(i,j in puntos)x(i,j)is_binary
forall(j in puntos)z(j)is_binary
objetivo:=sum(i,j in puntos) c(i,j)*x(i,j)+sum(j in puntos)v(j)*z(j)
res_1:=sum(j in puntos)z(j)=m
forall(j in puntos)res_2(j):=sum(i in puntos)dem(i)*x(i,j)<=cap
forall(i in puntos)res_3(i):=sum(j in puntos)x(i,j)=1
forall(i in puntos,j in puntos)res_4(i,j):=x(i,j)<=z(j)
!===============OPTIMIZACIÓN===============
starttime:=gettime
setparam("XPRS_MAXTIME",tiempo_calculo)
minimize(objetivo)
nvar:= getparam("XPRS_cols")
nres:= getparam("XPRS_rows")
writeln("\nNumero de variables: ",nvar,", numero de restricciones: ",nres)
mip_status:= getparam("XPRS_mipstatus")
mejor_cota:= getparam("XPRS_bestbound")
nodos_BB:=getparam("XPRS_nodes")
if(getobjval<>0)then
      hueco_final:= 100*(getobjval-mejor_cota)/getobjval
end-if
!================RESULTADOS================
writeln("\nSolucion del problema de clustering:")
writeln("mip_status = ",mip_status)
writeln("Valor objetivo:  ",getobjval)
writeln("Mejor cota= ",mejor_cota)
writeln("Hueco final= ",strfmt(hueco_final,6,2))
writeln("Nodos examinados = ",nodos_BB)
writeln("Tiempo en segundos = ",gettime-starttime)
writeln("\nClusters finales: distancia aproximada = ",getobjval,"\n\n")
ic:=0
forall(j in puntos)do
          if(z(j).sol>=0.99)then
                    ic:=ic+1
                    centro(ic):=j
                    npc(ic):=0
                    forall(i in puntos)do
                              if(x(i,j).sol>=0.99)then
                                        npc(ic):=npc(ic)+1
                                        ind(ic,npc(ic)):=i
                              end-if
                    end-do
          end-if
```

```
end-do
writeln
if(ic<>m)then writeln("El número de rutas y el de clusters no coinciden");end-if
if compile("tsp_tucker.mos")<>0 then
        writeln("Algún problema al compilar tsp_tucker.mos")
        exit(1)
end-if
load(mod_tsp, "tsp_tucker.bim")
dist_total:=0.0
forall(i in rutas)do
        forall(j in 1..npc(i))do
        indr(j):=ind(i,j)
end-do
nr:=npc(i)
initializations to "shmem:datos"
        n nr indr dist
        end-initializations
run(mod_tsp)
wait
dropnextevent
initializations from "raw:"
        dist_ruta as "shmem:dist_ruta"
end-initializations
writeln("dist_ruta = ",dist_ruta)
dist_total:=dist_total+dist_ruta
end-do
fdelete("tsp_tucker.bim")
writeln("\nHeurística de clustering de Bramel-Simchi Levi, distancia = ",dist_total)
writeln("Tiempo total en segundos = ",gettime-starttime)
end-model
```

Mejora exacta del Algoritmo de Ahorros mediante formulación exacta de Tucker- Miller- Zemlin

```
model " Heuristica de Clarke & Wright con formulación exacta de Tucker- Miller- Zemlin"
uses "mmxprs"
uses "mmsystem"
parameters
        archivo_datos =  "C:eilon_7_2.dat"
end-parameters
declarations
        n:integer
        m:integer
end-declarations
fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        dist:array(nodos,nodos) of integer
        dem:array(nodos)of integer
        cap:integer
        rutas = 1..n-1
        nnr:array(rutas)of integer
        indr:array(nodos,rutas)of integer
        ruta_activa:array(rutas)of integer
        carga:array(rutas)of integer
        ind:array(nodos,rutas)of integer
```

```
        nra:integer
        sig:array(nodos,rutas)of integer
        siga:array(nodos,rutas)of integer
        ini:array(rutas) of integer
        fin:array(rutas) of integer
        sav:array(nodos,nodos)of integer
        nsav:integer
        savord:array(range)of integer
        indi:array(range)of integer
        indj:array(range)of integer
        marcado:array(nodos,nodos)of integer
        dist_ruta:array(rutas)of integer
        distotal, daux:integer
        i1, final, iter, ruta, ic, jact, is, js, t1:integer
        inicial = 1
        tol = 0.001
        tini:real
end-declarations
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do
forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================DATOS==================
writeln("Problema de routing con capacidades  CVRP, \nn = ",n,", m = ",m)
writeln("Datos: ",archivo_datos)
writeln("\Heuristica de Clarke-Wright")
writeln("\nDemandas: ")
!forall(i in nodos)writeln(dem(i))
writeln("Demanda total: ",dem_total)
tini:=gettime
writeln("\nMatriz de distancias:\n")
forall(i in nodos)do
  writeln
  forall(j in nodos)write(dist(i,j),"\t")
end-do
writeln("\nCapacidad = ",cap)
writeln("Mínimo número de vehículos: ",ceil(dem_total/cap))
writeln("Número de vehículos: ",m)
!==========SOLUCIÓN INICIAL: n-1 rutas=========
nra:=0
dsol:=0
forall(i in nodos|i>1)do
        nra:=nra+1
        nnr(nra):=1
        sig(i,nra):=1
        sig(1,nra):=i
        ruta_activa(nra):=1
        carga(nra):=dem(i)
        ini(nra):=i
        fin(nra):=i
        dist_ruta(nra):=dist(1,i)+dist(i,1)
        dsol:=dsol+dist(1,i)+dist(i,1)
end-do
writeln("\nSolución inicial, dist = ",dsol)
!======CÁLCULO DE LA MATRIZ DE AHORROS=====
```

```
forall(i,j in nodos|i>1 and j>1)sav(i,j):=dist(i,1)+dist(1,j)-dist(i,j)
writeln("\nMatriz de ahorros:\n")
forall(j in nodos|j>1)write(j,"\t")
writeln
forall(i in nodos|i>1 )do
        write("\n",i,"\t")
        forall(j in nodos|j>1)write(sav(i,j),"\t")
end-do
!===LISTA ORDENADA DE AHORROS POSITIVOS===
nsav:=0
final:=0
while(final=0)do
        nsav:=nsav+1
        daux:=-999
        forall(i,j in nodos|i>1 and j>1 and marcado(i,j)=0)do
                if(sav(i,j)>daux)then
                        daux:=sav(i,j)
                        ik:=i
                        jk:=j
                end-if
        end-do
        marcado(ik,jk):=1
        savord(nsav):=daux
        indi(nsav):=ik
        indj(nsav):=jk
        if(daux<0)then final:=1;end-if
end-do
nsav:=nsav-1
writeln("\nSavings ordenados:\n")
writeln("t\tsav\ti\tj\n")
forall(t in 1..nsav)writeln(t,"\t",savord(t),"\t",indi(t),"\t",indj(t))
!==============BUCLE PRINCIPAL==============
forall(t in 1..nsav)do
        is:=indi(t)
        js:=indj(t)
        forall(k1, k2 in rutas|k1<>k2 and ruta_activa(k1)=1 and ruta_activa(k2)=1)do
                if(is=fin(k2) and js=ini(k1) and carga(k1)+carga(k2)<=cap)then
                        writeln("mezcla de las rutas ",k1," y ",k2,", is = ",is,", js = ",js,", saving =
                        ",savord(t))
                        ruta_activa(k1):=0
                        sig(is,k2):=js
                        jact:=js
                        while(jact<>fin(k1))do
                                sig(jact,k2):=sig(jact,k1)
                                jact:=sig(jact,k1)
                        end-do
                        fin(k2):=fin(k1)
                        sig(fin(k2),k2):=1
                        carga(k2):=carga(k1)+carga(k2)
                        nnr(k2):=nnr(k1)+nnr(k2)
                        nra:=nra-1
                        break
                end-if
        end-do
end-do
!================RESULTADOS================
writeln("\nSolución final de la heurística de Clarke and Wright:")
writeln("Número de rutas: ",nra)
nra:=0
```

```
distotal:=0  ! distancia total de la ruta
forall(k1 in rutas|ruta_activa(k1)=1)do
        nra:=nra+1
        write("\nruta ",k1,", carga: ",carga(k1),", ini = ",ini(k1),", fin = ",fin(k1),", nrecorrido: ")
        t1:=1;indr(t1,k1):=1
        write(1)
        jact:=ini(k1)
        t1:=t1+1
        indr(t1,k1):=jact
        write("-",jact)
        distotal:=distotal+dist(1,ini(k1))
        while(jact<>fin(k1))do
                distotal:=distotal+dist(jact, sig(jact,k1))
                jact:=sig(jact,k1)
                t1:=t1+1
                indr(t1,k1):=jact
                write("-",jact)
        end-do
        write("-",1)
        distotal:=distotal+dist(fin(k1),1)
        write(", nnr = ",nnr(k1)+1,", t1 = ",t1)
        nnr(k1):=t1
end-do
writeln("\n\nIndices de cada ruta:")
forall(k in rutas|ruta_activa(k)=1)do
        write("\nruta : ",k," => ")
        forall(t in 1..nnr(k))write(indr(t,k),",")
end-do
writeln("\n\nDistancia total = ",distotal)
writeln("Tiempo total = ",gettime-tini)
!================MEJORA================
if compile("tsp_tucker.mos")<>0 then
          writeln("Algún problema al compilar tsp_tucker.mos")
          exit(1)
end-if
load(mod_tsp, "tsp_tucker.bim")
writeln("\n\nMejora  de cada ruta con Tucker-Miller:")
tini:=gettime
distotal2:=0
forall(k in rutas|ruta_activa(k)=1)do
        write("\nruta : ",k," => ")
        forall(t in 1..nnr(k))indr2(t):=indr(t,k)
        nnr2:=nnr(k)
        initializations to "shmem:datos"
        n nnr2 indr2 dist
        end-initializations
        run(mod_tsp)
        wait
        dropnextevent
        initializations from "raw:"
                dist_mej as "shmem:dist_ruta"
        end-initializations
        writeln("dist_ruta mejorada= ",dist_mej)
        distotal2:=distotal2+dist_mej
end-do
writeln("\n\nDistancia total mejorada= ",distotal2)
writeln("Tiempo total en la mejora= ",gettime-tini)
end-model
```

Anexo 4: Programación de metaheurística GRASP aplicada al Algoritmo de Ahorros de Clarke & Wright

```
model " Heuristica GRASP basada en Clarke-Wright para CVRP"
uses "mmxprs"
uses "mmsystem"
uses "mmjobs"

parameters
        archivo_datos = "C:/eilon_7_2.dat"
end-parameters

declarations
        n:integer
        m:integer
end-declarations

fopen(archivo_datos,F_INPUT)
read(n)
read(m)
declarations
        nodos = 1..n
        dist:array(nodos,nodos) of integer
        dem:array(nodos)of integer
        cap:integer
        rutas = 1..n-1
        nnr:array(rutas)of integer
        indr:array(nodos,rutas)of integer
        indr2:array(nodos)of integer
        nnr2, distotal2:integer
        ruta_activa:array(rutas)of integer
        dist_ruta:array(rutas)of integer
        dist_mej:integer
        carga:array(rutas)of integer
        ind:array(nodos,rutas)of integer
        nra:integer
        sig:array(nodos,rutas)of integer
        siga:array(nodos,rutas)of integer
        ini:array(rutas) of integer
        fin:array(rutas) of integer
        sav:array(nodos,nodos)of integer
        nsav:integer
        savord:array(range)of integer
        indi:array(range)of integer
        indj:array(range)of integer
        marcado:array(nodos,nodos)of integer
        distotal, daux:integer
        N = 40
        K = 8
        indrcl:array(1..K)of integer
        mejora_tucker = 0
        i1, final, iter, ruta, ic, jact, is, js, t1:integer
        inicial = 1
        tol = 0.001
        tini,time1:real
        mod_tsp: Model
        salida_errores_tsp:string
end-declarations
```

```
forall(i in nodos)do
        read(i1)
        read(dem(i))
end-do

forall(i in nodos,j in nodos)read(dist(i,j))
read(cap)
fclose(F_INPUT)
dem_total:=sum(i in nodos|i<>1)dem(i)
!=================DATOS==================
writeln("Problema de routing con capacidades  CVRP, \nn = ",n,", m = ",m)
writeln("Datos: ",archivo_datos)
writeln("\Heuristica de Clarke-Wright")
writeln("\nDemandas: ")
!forall(i in nodos)writeln(dem(i))
writeln("Demanda total: ",dem_total)
tini:=gettime
writeln("\nMatriz de distancias:\n")
forall(i in nodos)do
        writeln
        forall(j in nodos)write(dist(i,j),"\t")
end-do
writeln("\nCapacidad = ",cap)
writeln("Mínimo número de vehículos: ",ceil(dem_total/cap))
writeln("Número de vehículos: ",m)
forall(i,j in nodos|i>1 and j>1)sav(i,j):=dist(i,1)+dist(1,j)-dist(i,j)
time1:=gettime
writeln("\nMatriz de ahorros:\n")
forall(j in nodos|j>1)write(j,"\t")
writeln
forall(i in nodos|i>1 )do
        write("\n",i,"\t")
        forall(j in nodos|j>1)write(sav(i,j),"\t")
end-do
!===LISTA ORDENADA DE AHORROS POSITIVOS===
nsav:=0
final:=0
while(final=0)do
        nsav:=nsav+1
        daux:=-999
        forall(i,j in nodos|i>1 and j>1 and marcado(i,j)=0)do
                if(sav(i,j)>daux)then
                        daux:=sav(i,j)
                        ik:=i
                        jk:=j
                end-if
        end-do
        marcado(ik,jk):=1
        savord(nsav):=daux
        indi(nsav):=ik
        indj(nsav):=jk
        if(daux<=0)then final:=1;end-if
end-do
nsav:=nsav-1
writeln("\nSavings ordenados:\n")
writeln("t\tsav\ti\tj\n")
forall(t in 1..nsav)
  writeln(t,"\t",savord(t),"\t",indi(t),"\t",indj(t))
```

```
mejor_dist:=99999
forall(itergrasp in 1..N)do
!==========SOLUCIÓN INICIAL: n-1 rutas=========
nra:=0
dsol:=0
forall(i in nodos|i>1)do
        nra:=nra+1
        nnr(nra):=1
        sig(i,nra):=1
        sig(1,nra):=i
        ruta_activa(nra):=1
        carga(nra):=dem(i)
        ini(nra):=i
        fin(nra):=i
        dist_ruta(nra):=dist(1,i)+dist(i,1)
        dsol:=dsol+dist(1,i)+dist(i,1)
end-do
writeln("\nIteracion ",itergrasp,", Solución inicial, dist = ",dsol)
!==========RCL inicial=========
forall(k in 1..K)indrcl(k):=k
ts:=K
final:=0
while(final=0)do
        kr:=ceil(random*K)
        is:=indi(indrcl(kr))
        js:=indj(indrcl(kr))
        writeln("sav(",is,",",js,") = ",savord(t))
        forall(k1, k2 in rutas|k1<>k2 and ruta_activa(k1)=1 and ruta_activa(k2)=1)do
                if(is=fin(k2) and js=ini(k1) and carga(k1)+carga(k2)<=cap)then
                        ruta_activa(k1):=0
                        sig(is,k2):=js
                        jact:=js
                        while(jact<>fin(k1))do
                        sig(jact,k2):=sig(jact,k1)
                        jact:=sig(jact,k1)
        end-do
        fin(k2):=fin(k1)
        sig(fin(k2),k2):=1
        carga(k2):=carga(k1)+carga(k2)
        nnr(k2):=nnr(k1)+nnr(k2)
        nra:=nra-1
        break
        end-if
end-do
ts:=ts+1
indrcl(kr):=ts
if(ts>nsav)then final:=1;end-if
end-do
!==========SOLUCIÓN FINAL=========
writeln("\nSolución final de la heurística de Clarke and Wright aleatorizada:")
writeln("Número de rutas: ",nra)
nra:=0
distotal:=0  ! distancia total de la ruta
distotal2:=0  ! distancia total de la ruta mejorada
forall(k1 in rutas|ruta_activa(k1)=1)do
        nra:=nra+1
        dist_ruta(k1):=0
        write("\nruta ",k1,", carga: ",carga(k1),", ini = ",ini(k1),", fin = ",fin(k1),", nrecorrido: ")
        t1:=1;indr(t1,k1):=1
```

```
            jact:=ini(k1)
            t1:=t1+1
            indr(t1,k1):=jact
            distotal:=distotal+dist(1,ini(k1))
            dist_ruta(k1):=dist_ruta(k1)+dist(1,ini(k1))
            while(jact<>fin(k1))do
                        distotal:=distotal+dist(jact, sig(jact,k1))
                        dist_ruta(k1):=dist_ruta(k1)+dist(jact, sig(jact,k1))
                        jact:=sig(jact,k1)
                        t1:=t1+1
                        indr(t1,k1):=jacT
            end-do
            distotal:=distotal+dist(fin(k1),1)
            dist_ruta(k1):=dist_ruta(k1)+dist(fin(k1),1)
            write(", nnr = ",nnr(k1)+1,", t1 = ",t1,", dist_ruta = ",dist_ruta(k1))
            nnr(k1):=t1
end-do
writeln("\nnra = ",nra)
writeln("\n\nIndices de cada ruta:")
forall(k in rutas|ruta_activa(k)=1)do
            write("\nruta : ",k," => ")
            forall(t in 1..nnr(k))write(indr(t,k),",")
            write("  dist = ",dist_ruta(k))
end-do
writeln("\n\nDistancia total = ",distotal)
writeln("Tiempo total = ",gettime-tini)
!==========MEJORA DE LAS RUTAS CON TUCKER- MILLER=========
if(mejora_tucker = 1)then
if compile("tsp_tucker.mos")<>0 then
              writeln("Algún problema al compilar tsp_tucker.mos")
              exit(1)
end-if
load(mod_tsp, "tsp_tucker.bim")
!writeln("\n\nMejora  de cada ruta con Tucker-Miller:")
tini:=gettime
distotal2:=0
forall(k in rutas|ruta_activa(k)=1)do
            write("\nruta : ",k," => ")
            forall(t in 1..nnr(k))indr2(t):=indr(t,k)
            nnr2:=nnr(k)
            initializations to "shmem:datos"
                        n nnr2 indr2 dist
            end-initializations
            run(mod_tsp)
            wait
            dropnextevent
            initializations from "raw:"
                        dist_mej as "shmem:dist_ruta"
            end-initializations
            writeln("dist_ruta mejorada= ",dist_mej)
            distotal2:=distotal2+dist_meJ
end-do
writeln("\n\nDistancia total mejorada= ",distotal2)
writeln("Tiempo total en la mejora= ",gettime-tini)
end-if
if(distotal<mejor_dist)then
            mejor_dist:=distotal
end-if
writeln("\n\nDistancia mejor final= ",mejor_dist)
```

```
writeln("Tiempo total = ",gettime-time1)
end-do
end-model
```

BIBLIOGRAFÍA

[1] "*A Comparison of Techniques for Solving the Fleet Size and Mix Vehicle Routing Problem".* F. Gheysens, B. Golden, A. Assad. (1984). Article (Operations Research Spektrum).

[2] *"A Guide to Vehicle Routing Heuristics".* J. F. Cordeau, M. Gendreau, G. Laporte, J. Y. Potvin, F. Semet. (2002). Article (Operational Research Society).

[3] *"A heuristic algorithm for the Asymmetric Capacitated Vehicle Routing Problem".* Daniele Vigo. (1994).

[4] "*A Heuristic Approach Based on Clarke- Wright Algorithm for Open Vehicle Routing Problem".* Tantikorn Pichpibul, Ruengsak Kawtummanchai. (2013). Article (The Scientific World Journal).

[5] "*Algoritmos Heurísticos en Optimización Combinatoria".* Rafael Martí. (2003). Artículo (Departamento de Estadística e Investigación Operativa, Universidad de Valencia).

[6] "*A location based heuristic for general routing problems".* Julien Bramel, Dabid Simchi- Levi. (1995). Article (Operations Research).

[7] *"An effective Approach to Routing Problems".* Ida Stankovianska. (2012). Article (Department of Mathematical Methods, University of Zilina).

[8] *"A new enhacement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem".* IK. Altinel, T. Öcan. (2004). Article (Operational Research Society).

[9] "A*n improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem".* Tantikorn Pichpibul, Ruengsak Kawtummanchai. (2012). Article (University of Thailand).

[10] "*Aplicación de la Metaheurística Búsqueda de la Armonía para Resolver el Problema de Ruteo de Vehículos con Inventarios".* Ramiro Santos Atiencia, Ricardo Aceves García. (2012). Artículo (Universidad Nacional Autónoma de México).

[11] *"A robust enhacement to the Clarke-Wright saving algorithm".* T. Doyuran, B. Catay. (2010). Article (Sabanci University).

[12] *"A Simple and Efficient Perturbation Heuristic to solve the Vehicle Routing Problem".* Jacques Renaud, Fayez F. Boctor. (2005). Article (Université Laval, Québec, Canada).

[13] *"Bio- inspired Algorithms for the Vehicle Routing Problem".* Francisco Baptista Pereira, Jorge Tavarés. (2009).

[14] *"Caracterización, Modelado y Determinación de las Rutas de la Flota en una Empresa de Rendering".* Jorge Rodríguez Pérez. Trabajo fin de Máster.

[15] *"Clarke and Wright Algorithm".* M. Battarra, R. Baldacci, D. Vigo. (2007) Università di Bologna.

[16] "*Clarke & Wright´s Savings Algorithm".* Jens Lysgaard. (1997). Article (Department of Management Sciencie and Logistics. The Aarhus School of Business).

[17] "*Cooperación en los problemas del viajante (TSP) y de rutas de vehículos (VRP): una panorámica".* A. Calvino Martínez. (2011).

[9] "*CVRP Flota heterogénea".* J. Sáez Aguado. (2014). Apuntes de clase.

[10] "*Desarrollo de un algoritmo heurístico para establecer las rutas de transporte escolar de la secretaría de educación de Bogotá".* Wilson Nicolas Barajas Mora. (2009). Artículo (Universidad Nacional de Colombia)

[11] "*El problema TSP".* J. Sáez Aguado. (2014). Apuntes de clase.

[12] "*El problema de rutas de vehículos. Extensiones y métodos de resolución, estado del arte".* Armin Lüer, Magdalena Benavente, Jaime Bustos, Bárbara Venegas. (2009). Artículo (Universidad de La Frontera, Chile).

[13] *"Fleet management and logistics".* Teodor Gabriel Crainic, Gilbert Laporte. (1998).

[14] *"Handbook of metaheuristics".* Fred Glover, Gary A. Kochenberger. (2003).

[15] "*Heurísticas para Problemas de Ruteo de Vehículos".* Alfredo Olivera. (2004). Artículo (Universidad de la República).

[16] *"Improving parametric Clarke and Wright algorithms by means of iterative empirically adjusted greedy heuristics".* Albert Corominas, Alberto García- Villoria, Rafael Pastor. (2014). Article (Institute of Industrial and Control Engineering).

[17] *"Métodos Exactos y Heurísticos para resolver el Problema del Agente Viajero (TSP) y el Problema de Ruteo de Vehículos (VRP)".* Fernando Sandoya Sánchez. (2007). Artículo (Escuela Superior Politécnica del Litoral, Ecuador).

[18] *"Models, relaxations and exact approaches for the capacitated vehicle routing problem"*. Paolo Toth, Daniele Vigo. (2000). Article (Discrete Applied Mathematics).

[19] *"Network Routing".* M.O. Ball, T.L. Magnanti, C.L. Monm, G.L. Nemhauser. Article (Handbooks in Operations Research and Management Science).

[20] *"On the Capacitated Vehicle Routing Problem".* T. K. Ralpsht, L. Kopmant, W.R. Pulleyblanks, L. E. Trotter. (2001). Article (Mathematical Programming).

[21] "*Optimización en la entrega de productos para una cadena de abastecimientos".* Flores Torres, Mario Alfredo. Tesis.

[22] J. Sáez Aguado. (2014). "*Problemas de rutas de vehículos".* Apuntes de clase.

[23] *"Routing a Heterogeneous Fleet of Vehicles"*. Roberto Baldacci, Maria Battarra, Daniele Vigo. (2007).

[24] *"Scheduling and Routing in transportation and distribution systems: formulations and new relaxations".* B. Gavish, S. C. Graves. (1981).

[25] "*Scheduling of vehicles from a central depot to a number of delivery points".* G. Clarke, W. Wright. (1964). Article.

[26] "*The fleet size and mix vehicle routing problem".* B. Golden, A. Assad, L. Levys, F. Gheysens. (1984). Article (Operations Research Spektrum).

[27] *"The life and times of the Saving Method for Vehicle Routing Problem".* Graham K. Rand. (2009). Article (The Journal of ORSSA).

[28] *"The periodic vehicle routing problema: a case study"*. S. Coene, A. Arnout, F. Spieksma. (2008). Article (Department of Decision Sciences and Information Management (KBI)).

[29] *"The savings algorith for the vehicle routing problem".* H. Paessens. (1988). Article (European Journal of Operational Research).

[30] "*The Vehicle Routing Problem".* Paolo Toth, Daniele Vigo. (2002). Ed. Siam.

[31] "*Traveling Salesman Problem, Theory and Applications".* Donald Davendra (2010).

[32] *"Tuning a Parametric Clarke- Wright Heuristic via Genetic Algorithm"*. Maria Battarra, Bruce Golden, Daniele Vigo. (2006). Article (Journal of the Operational Research Society).

[33] "*Una revisión al estado del arte del problema de ruteo de vehículos. Evolución histórica y métodos de solución".* L. Rocha, C. González, J. Orijuela. (2011). Artículo (Universidad Distrital Francisco José de Caldas).

[34] *"Vehicle Routing".* Jean- François Cordeau, Gilbert Laporte, Martin W. P. Savelsbergh, Daniele Vigo (2005). Article (GERARD).

[35] *"Vehicle Routing and Scheduling".* Martin Savelsbergh. Article (Georgia Institute of Technology).

[36] *"Vehicle Routing: Methods and Studies".* Bruce L. Golden, Arjang Assad. (1988).

[37] http://comopt.ifi.uniheidelberg.de/software/TSPLIB95/vrp/.
Fecha de consulta: marzo, 2015.

[38]http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html.
Fecha de consulta: marzo, 2015.

[39] http://neo.lcc.uma.es/vrp/vrp-flavors/vrp-with-backhauls/.
Fecha de consulta: abril, 2015.

[40] http://www.systeam.com.ar/.

Fecha de consulta: abril, 2015.

[41] http://www.routingreparto.com/.

Fecha de consulta: abril, 2015.

[42] http://personales.upv.es/arodrigu/rutas/index.htm.

Fecha de consulta: marzo, 2015.

[43] http://www.fico.com/en/products/fico-xpress-optimization-suite.

Fecha de consulta: marzo, 2015.

[44] http://www.orms-today.org/surveys/Vehicle_Routing/vrss.html

Fecha de consulta: marzo, 2015.